UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Infra-estrutura de Software Baseada em Componentes para a Construção de Aplicações para Comunidades Virtuais Móveis

Glauber Vinícius Ventura de Melo Ferreira

Campina Grande, Paraíba, Brasil
Outubro de 2006

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Infra-estrutura de Software Baseada em Componentes para a Construção de Aplicações para Comunidades Virtuais Móveis

Glauber Vinícius Ventura de Melo Ferreira

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande – Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação (MSc).

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Modelos Computacionais e Cognitivos

Angelo Perkusich
Orientador
Evandro de Barros Costa
Orientador

Campina Grande, Paraíba, Brasil
Outubro de 2006

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

F383i Ferreira, Glauber Vinícius Ventura de Melo

2006 Infra-estrutura de software baseada em componentes para a construção de aplicações para comunidades virtuais móveis/ Glauber Vinícius Ventura de Melo Ferreira. – Campina Grande, 2006

79fs.: il.

Referências

Dissertação (Mestrado em Informática) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Orientadores: Angelo Perkusich e Evandro de Barros Costa

1– Palm-top 2– Comunidades Virtuais 3– Componentes I-Título

CDU 004.382.75

Resumo

A popularização do uso de dispositivos móveis e tecnologias de comunicação sem fio tem permitido o desenvolvimento de diversas aplicações relacionadas com comunidades virtuais móveis. Alguns exemplos de tais aplicações são: *mobile learning*, gerenciamento de *workflow* e *healthcare communities*. Estas aplicações possuem características comuns, tais como o estabelecimento de comunidades, a notificação da proximidade física de dispositivos e indivíduos, a representação dos interesses e das preferências do indivíduo, a disponibilização de conteúdo para os membros das comunidades, entre outras.

Neste trabalho apresenta-se uma infra-estrutura de software baseada em componentes para a construção de aplicações para comunidades virtuais móveis, abordando as características mencionadas anteriormente. Esta infra-estrutura possibilita o desenvolvimento sistemático de aplicações neste contexto, com base nos conceitos de reutilização e flexibilidade. Como forma de validar a infra-estrutura desenvolvida, apresenta-se o protótipo de uma aplicação para uma comunidade virtual móvel.

Abstract

The popularization of mobile devices and wireless communication technologies has allowed the development of various applications concerning mobile virtual communities. Examples of such applications are related to mobile learning, workflow management, and healthcare communities. These applications have some issues in common, such as establishment of communities, notification of proximity among individuals, representation of individuals' interests, share of content among the communities' members, among others.

In this work we introduce a component-based software infrastructure for building mobile virtual communities applications dealing with the above mentioned issues. This infrastructure allows a systematic development of applications in this context, based on the concepts of reusability and flexibility. In order to validate the developed infrastructure, we present a prototype application for mobile virtual communities.

Agradecimentos

Agradeço aos professores que me orientaram durante este trabalho: Prof. Dr. Angelo Perkusich e Prof. Dr. Evandro Costa. Suas críticas construtivas e sugestões providenciais foram primordiais para a existência deste trabalho.

Agradeço às outras pessoas que também contribuíram para a execução deste trabalho: Hyggo Almeida, Emerson Loureiro, Frederico Bublitz, Loreno Oliveira e Olympio Cipriano.

Agradeço às conversas realizadas nas "reuniões de trabalho" no Restaurante Miúra. Aprendi bastante com as pessoas que delas participaram e esse aprendizado me ajudou durante o andamento deste trabalho.

Agradeço a todos os meus familiares que me deram apoio durante o desenvolvimento deste trabalho. As perguntas repetitivas sobre o que era mestrado, quando eu acabaria o mestrado, e se eu já estava trabalhando, estão perdoadas.

Agradeço aos meus amigos de Maceió, que me faziam esquecer um pouco do trabalho em Campina Grande, sempre que eu estava na cidade ímpar: Marcelo Henrique, Rodrigo Antônio, Geraldo Sitônio, Paulo Henrique, Paulo Marcos, Elton Felipe, Márcio Oliveira e Alan Pedro. O restante está incluso no penúltimo parágrafo.

Agradeço aos companheiros de apartamento, Emerson Loureiro e Elthon Allex, com os quais convivi durante o mestrado. Que ambos tenham sucesso em suas novas direções.

Agradeço às quatro pessoas que me fizeram evoluir como profissional e pessoa, durante o mestrado: Hyggo Almeida, Marcelo Henrique, Angelo Perkusich e Leandro Dias.

Agradeço aos novos amigos e amigas que fiz no Laboratório de Sistemas Embarcados e Computação Pervasiva, no Residencial Flamingo, nas viagens para João Pessoa e Garanhuns, e em Campina Grande. Cada um do seu jeito, me fizeram aprender alguma coisa.

Como não poderia deixar de esquecer dos esquecimentos da minha memória, agradeço também a todas pessoas que contribuíram indiretamente para a execução deste trabalho e cujos nomes foram esquecidos.

Por fim, agradeço a Deus, que me deu a chance de poder chegar até esse momento e fazer estes agradecimentos verdadeiros.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Algoritmos

Capítulo 1

Introdução

O estabelecimento de relações coletivas é uma característica inerente ao indivíduo, uma vez que viver de forma isolada não faz parte da sua natureza. Essa particularidade do ser humano é uma das razões pelas quais os indivíduos sempre têm se organizado em comunidades nas quais mantêm relações com outros indivíduos, os quais geralmente vivem em uma mesma área. Por exemplo, desde o período pré-histórico, os povos nômades que habitavam regiões geográficas próximas já se organizavam em comunidades, buscando desta forma facilitar a sobrevivência através da cooperação entre os indivíduos. Pode-se observar então que a proximidade física entre os indivíduos é uma das características que motivam a constituição de comunidades.

Por outro lado, comunidades são também constituídas devido ao compartilhamento de interesses comuns por parte dos seus membros. Alguns exemplos de comunidades constituídas dessa forma são: comunidades religiosas (comunidade dos beneditinos, comunidade evangélica, comunidade carismática, etc.), comunidades formadas por residentes em um país estrangeiro (comunidade brasileira nos Estados Unidos, comunidade japonesa em São Paulo, etc.), comunidades relacionadas a uma determinada profissão (médicos, cientistas, advogados, etc.), entre outras [Hou01]. Em todos esses casos pode existir um distanciamento físico entre os indivíduos. Porém, isso não impede a formação de comunidades, uma vez que estes indivíduos possuem interesses em comum. Mesmo vivendo em áreas diferentes, os membros dessas comunidades realizam reuniões e encontros periódicos, estabelecendo desta forma relações coletivas.

A popularização da Internet na década de 1990, aliada à já estabelecida utilização dos

computadores pessoais, permitiu a criação de novos tipos de comunidades. Através das denominadas *Comunidades Virtuais* [Rhe93], os indivíduos podem se comunicar utilizando-se de ferramentas, tais como e-mails, fóruns eletrônicos, programas de mensagem instantânea e sessões de videoconferência. Com isso, a distância física entre os indivíduos deixou realmente de ser uma limitação para a constituição de comunidades. Pessoas vivendo em países diferentes passaram a utilizar computadores pessoais com acesso à Internet para interagir e se comunicar. Houve então a disseminação de comunidades com foco no ensino à distância, nas quais os aprendizes podem estar localizados em qualquer parte do mundo; comunidades relacionadas a software, formadas por usuários e desenvolvedores espalhados pelo mundo, como por exemplo comunidades de usuários Linux e comunidades de desenvolvedores Java; entre outras.

Os recentes avanços nas tecnologias de comunicação sem fio e na fabricação de dispositivos móveis possibilitaram a criação de um novo tipo de comunidade: as *Comunidades Virtuais Móveis*. Através de dispositivos móveis interconectados, indivíduos podem trocar informações e conhecimento, em qualquer lugar, a qualquer hora. Por exemplo, um indivíduo portando um *smartphone* pode dar continuidade ao seu curso à distância sobre planejamento financeiro, durante seu trajeto de ônibus do trabalho para casa. Ele pode acessar todo o material didático disponível na comunidade "Planejamento Financeiro", assim como interagir com os outros membros da comunidade que também estejam conectados, através de ferramentas de mensagem instantânea.

1.1 Problemática

Assim como em outros domínios de aplicação, o desenvolvimento de software para Comunidades Virtuais Móveis envolve características que são comuns à maioria das aplicações deste domínio. É necessário representar os interesses do indivíduo assim como implementar algoritmos para identificar indivíduos com interesses comuns; permitir a formação de comunidades; notificar a proximidade física de indivíduos; definir níveis de acesso às informações disponíveis nas comunidades; entre outras características.

Embora alguns trabalhos já tenham sido propostos para facilitar o desenvolvimento desse tipo de aplicação, a maioria deles torna transparente ao desenvolvedor somente mecanismos

relacionados com a infra-estrutura de redes *ad hoc* [BCGS04], na qual as aplicações para Comunidades Virtuais Móveis são implantadas. Atualmente não existem abordagens que integrem as características comuns a esse domínio. É necessário reimplementar essas características para cada nova aplicação para Comunidades Virtuais Móveis.

1.2 Objetivo do Trabalho

O objetivo neste trabalho é prover uma infra-estrutura de software baseada em componentes para a construção de aplicações para Comunidades Virtuais Móveis, disponibilizando serviços que fazem parte do contexto de tais aplicações. Essa infra-estrutura deve disponibilizar mecanismos para o desenvolvimento de software de forma flexível, considerando características comuns às Comunidades Virtuais Móveis.

Essas características são implementadas como serviços providos por componentes de software, os quais interagem entre si para prover funcionalidades necessárias ao desenvolvimento de aplicações para Comunidades Virtuais Móveis, tais como criação de comunidades, notificação da aproximação de indivíduos, autorização de acesso às informações disponibilizados pelas comunidades, entre outras.

Como forma de validar os serviços dessa infra-estrutura, neste trabalho também é apresentada a definição e implementação de um protótipo do estudo de caso de uma aplicação para uma Comunidade Virtual Móvel implantada no Laboratório de Sistemas Embarcados e Computação Pervasiva da Universidade Federal de Campina Grande (`http://embedded.ufcg.edu.br/`).

1.3 Relevância

Aplicações para Comunidades Virtuais Móveis têm sido desenvolvidas em diversos domínios, tais como *mobile learning* [GK04], assistentes pessoais em conferências acadêmicas [SM02], gerenciamento de *workflow* [DCC03], aplicações que permitem a comunicação entre indivíduos próximos a uma determinada região [GSB+04], *healthcare communities* [KADL02], entre outros. Embora existam vários domínios nos quais os conceitos de Comunidades Virtuais Móveis estejam sendo aplicados, o desenvolvimento das

aplicações tem sido realizado de maneira não-sistemática, exigindo esforço desnecessário do desenvolvedor ao reimplementar, em cada nova aplicação, características comuns às Comunidades Virtuais Móveis.

A partir dos serviços disponibilizados pela infra-estrutura de software apresentada neste trabalho, os esforços realizados no desenvolvimento de aplicações para Comunidades Virtuais Móveis são direcionados aos serviços específicos do domínio da aplicação em questão. Aumenta-se então a produtividade na construção dos sistemas através da redução do tempo de desenvolvimento, uma vez que componentes podem ser reutilizados e configurados de acordo com o domínio de aplicação.

1.4 Estrutura da Dissertação

O restante deste trabalho está organizado como descrito a seguir:

- No Capítulo 2 é apresentada a fundamentação teórica sobre Comunidades Virtuais Móveis, com destaque para as características relacionadas com o conceito de mobilidade. As semelhanças e diferenças entre Comunidades Virtuais e Comunidades Virtuais Móveis são apresentadas, assim como alguns domínios de aplicações para Comunidades Virtuais Móveis. São apresentados também alguns trabalhos relacionados com a construção de aplicações para Comunidades Virtuais Móveis;

- No Capítulo 3 são descritos tópicos relacionados com software baseado em componentes, necessários para o entendimento deste trabalho. São apresentadas definições sobre reutilização, flexibilidade, disponibilização e configuração de componentes, composição dinâmica, evolução de software não antecipada, entre outras;

- No Capítulo 4 são apresentados os componentes especificados e implementados para facilitar a construção de aplicações para Comunidades Virtuais Móveis. Em cada uma das seções deste capítulo, as quais descrevem os componentes desenvolvidos neste trabalho, também são apresentados os serviços disponibilizados por cada um dos componentes;

- No Capítulo 5, como forma de validar os componentes desenvolvidos, são discutidas a definição e a implementação do protótipo do estudo de caso de uma aplicação para

uma Comunidade Virtual Móvel implantada no Laboratório de Sistemas Embarcados e Computação Pervasiva da Universidade Federal de Campina Grande;

- Por fim, no Capítulo 6 são apresentadas as conclusões do trabalho, apresentando as suas contribuições e discussões sobre trabalhos futuros.

No Apêndice A são apresentadas as especificações detalhadas dos componentes que constituem a infra-estrutura de software para a construção de aplicações para Comunidades Virtuais Móveis. No Apêndice B é apresentada a especificação detalhada do componente desenvolvido especificamente para uma aplicação de Comunidade Virtual Móvel.

Capítulo 2

Comunidades Virtuais Móveis

Rheingold define em seu livro *The Virtual Community* [Rhe93], que comunidades virtuais são grupos sociais mediados por computador. Tais comunidades promovem o crescimento do estabelecimento de relações coletivas entre indivíduos, uma vez que a mediação computacional das interações permite a formação de comunidades constituídas por pessoas localizadas em áreas geográficas distantes. Para auxiliar a interação entre os integrantes destas comunidades, várias ferramentas computacionais são utilizadas: fóruns de discussão, e-mail, *whiteboard*, sessões de audioconferência e videoconferência, mensagem instantânea, entre outras.

Em um dos seus artigos [Rhe03], Rheingold enumera algumas características das comunidades virtuais, definindo-as como:

1. Meios de comunicação muitos-para-muitos. Neste caso as comunidades virtuais possibilitam que muitas pessoas se comuniquem com muitas outras, diferente dos meios de comunicação um-para-muitos (*broadcast*) ou um-para-um (telefonia tradicional);

2. Organizadas por afinidades e interesses comuns, unindo pessoas que não necessariamente se conhecem antes de se encontrarem *online*;

3. Baseadas em texto, evoluindo para comunicação baseada em texto mais gráficos. Por décadas, as comunidades *online* utilizavam apenas texto não formatado. Os meios de comunicação baseados na Web possuem gráficos, animações, vídeo, sons, texto formatado;

4. Relativamente desassociadas da vida social face a face das comunidades geográficas. Pessoas que se comunicam ao redor do mundo sobre interesses comuns, muitas vezes não vivem próximas o suficiente para se encontrarem face a face regularmente.

É importante ressaltar a importância do item 2 na caracterização das comunidades virtuais, uma vez que a inexistência de interesses comuns entre os participantes inviabiliza a constituição dessas comunidades. A similaridade entre as preferências dos indivíduos é um fator importante no processo de estabelecimento desses grupos.

2.1 Comunidades Virtuais Evoluem com a Mobilidade

A presença constante dos mais variados tipos de dispositivos computacionais portáteis no cotidiano dos indivíduos é inegável: de celulares a notebooks, passando por *handhelds*, *smartphones* e *tablet pcs*. Todos esses dispositivos permitem a conectividade dos seus usuários através de tecnologias sem fio, tais como infravermelho, Wi-Fi, GPRS, WAP e Bluetooth. Esse cenário de mobilidade com conectividade tem promovido o aumento do estabelecimento de interações entre os indivíduos, possibilitando o surgimento das Comunidades Virtuais Móveis.

Fremaux considera as comunidades móveis como a evolução natural das comunidades virtuais, podendo ser vistas como comunidades virtuais nas quais serviços móveis são adicionados [Fre00]. A seguir são apresentados alguns aspectos que diferenciam as comunidades móveis das comunidades virtuais "tradicionais" baseadas na Web [FTF03]:

- Comunidades móveis podem ser acessadas através de dispositivos móveis como celulares, *smartphones* e PDAs. Isto pode levar a uma comunicação mais corriqueira entre os membros das comunidades, uma vez que esses dispositivos fazem parte do cotidiano dos indivíduos;

- Plataformas de comunidades móveis oferecem serviços diferenciados de comunicação para seus usuários: acesso ubíquo, possibilitando a conexão às suas comunidades em qualquer lugar a qualquer momento; e serviços baseados em localização, através da utilização de tecnologias de posicionamento [HSK04], tais como infravermelho, Bluetooth, GPS e Wi-Fi.

2.2 Características das Comunidades Virtuais Móveis

No mesmo artigo em que cita algumas características das Comunidades Virtuais, Rheingold define que as Comunidades Virtuais Móveis são [Rhe03]:

- Muitos-para-muitos, desktop e móvel, *always on*. Comunidades virtuais móveis e os recursos da Internet estão instantaneamente disponíveis para as pessoas e seus agentes de software em qualquer lugar que as pessoas estejam localizadas — em suas mesas de trabalho, no trânsito, em casa;

- Utilizadas para coordenar ações de grupos em espaços geográficos — grupos de adolescentes em shopping centers, ativistas mobilizados nas ruas;

Mais duas outras características presentes nas aplicações de Comunidades Virtuais Móveis podem ser citadas.

1. As aplicações de Comunidades Virtuais Móveis são acessadas através de uma grande *variedade de dispositivos* computacionais, com diferentes configurações de memória, capacidade de processamento e resolução gráfica. Isso faz com que durante o desenvolvimento de software para esse domínio seja considerada a diversidade de plataformas nas quais as aplicações são implantadas.

2. Este problema aparente é minimizado por meio da utilização de *informações contextuais* acerca do contexto no qual os indivíduos estão situados. Através dessas informações, as aplicações podem ser adaptadas com base nas preferências do usuário, nas configurações específicas do dispositivo utilizado e nas informações sobre a localização do usuário.

2.3 Domínios de Aplicação

Nesta seção são apresentados alguns domínios nos quais os conceitos de Comunidades Virtuais Móveis têm sido aplicados. São destacadas as características peculiares, de cada um dos domínios, que justificam a utilização dos conceitos de Comunidades Virtuais Móveis no desenvolvimento de aplicações. Além disso, também são descritos alguns benefícios obtidos com a utilização de aplicações de Comunidades Virtuais Móveis nestes domínios.

Healthcare Communities

Healthcare pode ser definido como a prevenção/tratamento de doenças, e a preservação do bem-estar físico e mental através de serviços oferecidos por médicos e outros profissionais de saúde [Lex06]. Além desses profissionais, algumas instituições também estão envolvidas na prestação desses serviços, tais como hospitais, organizações não-governamentais, companhias de seguro, entre outras [LDK02]. Os pacientes são os principais usuários desses serviços; eles interagem com os profissionais de saúde e com as instituições envolvidas nos sistemas de *healthcare*.

De acordo com algumas pesquisas realizadas, a demanda dos pacientes por informação aumenta quando recebem o resultado do diagnóstico de uma doença ou durante a realização de um novo tratamento [SCG99]. Outro fato identificado é a necessidade de comunicação com outro pacientes, com o intuito de trocar experiências e obter suporte emocional, principalmente quando são acometidos por doenças graves.

Um dos meios utilizados pelos pacientes para satisfazer suas necessidades de obtenção de informação e estabelecimento de interação são os grupos de auto-ajuda. Tais grupos são exemplos de comunidades, uma vez que seus membros estabelecem relações entre si e compartilham os mesmos interesses — a discussão sobre a doença em questão.

Embora a participação dos pacientes nesses grupos seja importante para auxiliar o tratamento da doença, alguns problemas podem fazer com que esses pacientes não se integrem nessas comunidades. Um desses problemas é a possível incompatibilidade de horários entre a agenda dos pacientes e as reuniões realizadas pelos grupos de auto-ajuda. Outro possível problema está relacionado com a dificuldade de locomoção que pode existir para que o paciente se desloque até os locais onde os membros do grupo se reúnem [LDK02].

Esses problemas citados anteriormente são solucionados quando os pacientes estabelecem relações coletivas através de aplicações para Comunidades Virtuais Móveis, uma vez que podem se comunicar a qualquer momento, independentemente das suas localizações. Além da superação desses problemas, a utilização dessas aplicações possibilita que os pacientes recebam informações contextuais, de acordo com a sua localização, tais como o endereço da farmácia e do médico mais próximos, e a notificação sobre a presença de outros pacientes localizados em uma região próxima. É importante ressaltar que o estabelecimento dessas comunidades não deve substituir o grupo de auto-ajuda propriamente dito; mas sim,

deve auxiliá-lo através da expansão dos mecanismos utilizados para interação.

Mobile Learning

Os recentes avanços obtidos na fabricação de dispositivos móveis sem fio possibilitaram a expansão dos mecanismos já utilizados para ensino/aprendizagem à distância. Através de aplicações de *mobile learning*, indivíduos que compartilham o interesse por aprender determinado assunto podem estabelecer comunidades nas quais interagem para obter conhecimento, independentemente das suas localizações, seja em um ônibus, na sala de espera de um consultório médico ou na fila de um banco.

Abordagens comuns de ensino/aprendizagem à distância, tais como universidades corporativas e cursos de graduação à distância, podem usufruir dos benefícios oferecidos pelas aplicações de *mobile learning*. Através da aplicação de conceitos de mobilidade às comunidades de aprendizado constituídas nas universidades corporativas, é possível que o treinamento de funcionários seja realizado não apenas nas dependências da empresa, mas também durante as atividades realizadas fora da empresa. Já os cursos de graduação à distância, freqüentados por pessoas com dificuldade em se adequar aos horários pré-determinados das instituições de ensino, podem utilizar os conceitos de mobilidade para permitir que os alunos desempenhem as atividades de aprendizagem independentemente de onde estejam localizados.

As atividades de aprendizado tradicionais também podem usufruir das características das aplicações de *mobile learning*. O cenário descrito a seguir, apresentado por Bryan Alexander [Ale04], ilustra algumas experiências possíveis. Imagine um estudante que assiste ao filme "Mestre dos Mares" e demonstra interesse pelo mundo da navegação do século XVIII. Sem nenhum direcionamento, o estudante pode buscar na *Amazon.com* por outros romances de Patrick O'Brian, assistir a um programa do *History Channel* sobre navegação, ou consultar o *Google.com* por páginas relacionadas. Em vez disso, a instituição de ensino pode configurar um ambiente no qual o estudante descobre que um professor de história ensina regularmente "a grande era da navegação" em várias turmas e possui páginas Web sobre as guerras navais da década de 1970; que a biblioteca possui recursos digitais e impressos sobre o tema; que alguns outros estudantes também possuem essa curiosidade e discutem sobre ela através de ferramentas de mensagem instantânea; e que um funcionário da instituição

navegou em um navio reconstruído do século XVIII no último verão e ficaria contente em discutir a experiência.

Embora todas essas experiências enriqueçam as atividades de aprendizado, é também importante ressaltar que as aplicações de *mobile learning* não devem substituir as atividades de ensino presencial. É tanto que, mesmo nos cursos de graduação à distância, uma determinada quantidade de aulas ocorre necessariamente em encontros presenciais.

Gerenciamento de *Workflow*

Sistemas de gerenciamento de *workflow* são freqüentemente utilizados para modelar, monitorar e controlar a execução coordenada de atividades executadas em diversos contextos [DCC03]. Este domínio de aplicação é caracterizado por uma interação freqüente entre os membros da equipe responsável pelas atividades, uma vez que os indivíduos precisam colaborar para que as atividades sejam executadas.

A diminuição dos custos associados à aquisição de dispositivos móveis tem feito com que o acompanhamento dessas atividades possa ser realizado diretamente no local onde são executadas. Isso possibilita que as tarefas possam ser coordenadas mesmo em equipes que estejam localizadas em áreas distintas e cujos membros envolvidos na execução das atividades se movimentam constantemente.

A seguir é descrito um sistema de gerenciamento de *workflow* que auxilia a execução de planos de emergência em uma companhia de petróleo, uma companhia de oleodutos e companhias de distribuição de gás [DCC03]. Nesse sistema, dispositivos móveis são utilizados pelo membros da equipe de emergência para registrar a execução de operações, permitindo que outras operações dependentes possam ser executadas mediante o término de uma operação principal. Por exemplo, diante de um caso de derramamento de petróleo, barreiras de contenção devem ser utilizadas, a extração de petróleo deve ser suspensa, procedimentos de limpeza devem ser executados, além do envolvimento de brigadas de incêndio nos casos mais sérios. Além do auxílio na coordenação das atividades, os membros da equipe têm acesso a várias informações necessárias para a execução das operações, tais como os materiais que devem ser utilizados na execução de cada procedimento, mapas, listas de autoridades que devem ser contactadas de acordo com a gravidade da emergência, entre outras.

2.4 Trabalhos Relacionados

Assim como em outros domínios de aplicação, o desenvolvimento de software para Comunidades Virtuais Móveis envolve características que são comuns à maioria das aplicações. Algumas abordagens têm sido propostas para auxiliar a construção de aplicações nesse contexto, evitando a implementação de todas as características presentes nas aplicações. Existem propostas que disponibilizam bibliotecas, arquiteturas, modelos de especificação, entre outras. A seguir são apresentadas as abordagens relacionadas ao trabalho apresentado neste documento.

Em [RLZ00] é apresentado um modelo de alto nível para a modelagem de sistemas móveis e distribuídos, denominado comunidades virtuais móveis. A modelagem dos sistemas é realizada através da abordagem multi-agentes [Wei00]. Este modelo utiliza conceitos definidos pelo *Reference Model for Open Distributed Processing* (RM-ODP) [Int02]. Comunidades móveis são modeladas através da composição de papéis criados para satisfazer um objetivo. Este objetivo é expressado através de políticas que são definidas por um conjunto de regras relacionadas às atividades desempenhadas na comunidade. Os papéis que constituem a comunidade são descritos por meio de interfaces de componentes que definem as interações das comunidades. O objetivo principal deste trabalho está na modelagem e especificação de comunidades virtuais móveis.

MOOsburg [CRI+01] é um ambiente colaborativo orientado a comunidades que modela a cidade de Blacksburg, EUA. Este trabalho disponibiliza um conjunto de ferramentas colaborativas que provêem acesso a conteúdos compartilhados, tais como *whiteboards* e *message boards*. Em 2002, foi proposta uma extensão deste trabalho, denominada *MOOsburg++* [FICR02], com o objetivo de dar suporte para comunidades virtuais sem fio. Esta extensão possibilita o acesso a partir de dispositivos móveis, tais como telefones celulares, *pagers* e PDAs. *MOOsburg++* provê interação síncrona e assíncrona com pessoas e dados. Os dados do ambiente (pessoas, lugares, coisas e objetos colaborativos) são representados por objetos Java e replicados para todos os clientes interessados. Para isso é utilizado o *Content Object Replication Kit* (CORK) [IRC01], um *toolkit* para a construção de aplicações distribuídas interativas, acessíveis através da Web.

ToothAgent [BGF05] é um protótipo de um sistema multi-agentes para suporte a

comunidades virtuais. Esse trabalho propõe uma arquitetura com servidores independentes nos quais plataformas multi-agentes podem ser instaladas e agentes podem agir representando seus usuários. Cada servidor provê serviços relacionados à área geográfica no qual está localizado; por exemplo, um servidor situado em uma universidade pode oferecer serviços de compra e venda de livros-texto. Os usuários podem contatar seus agentes pessoais utilizando telefones móveis ou PDAs com tecnologia Bluetooth. A arquitetura do *ToothAgent* é independente do domínio de aplicação uma vez que não depende de serviços específicos disponibilizados pelo servidor.

MyNet [1] é um projeto de colaboração entre o *Nokia Pervasive Computing Group* e a equipe do *MIT User Information Architecture*. Eles acreditam que a proliferação do uso de dispositivos pessoais com suporte para comunicação em rede demandará facilidade de uso, serviços de segurança, e conectividade ponto-a-ponto. Entre outros objetivos deste projeto, podem ser destacadas funcionalidades tais como: autorização e controle de acesso; integração de *middlewares* distribuídos e protocolos de descoberta de serviços; mecanismos intuitivos para "apresentar" um dispositivo em uma rede pessoal e para conceder acesso limitado para os recursos da rede pessoal a outros usuários; e composição de serviços e dispositivos.

Alguns outros trabalhos utilizam a abordagem multi-agentes para auxiliar o desenvolvimento de aplicações para Comunidades Virtuais Móveis. *Shine* [YKOK03] provê uma arquitetura P2P com agentes pessoais cujo foco está na comunicação entre os agentes que representam os usuários. *Proem* [KSP+01] é uma plataforma P2P para redes *ad-hoc* móveis que possibilita o desenvolvimento de aplicações colaborativas. No trabalho de Aldunate et. al. [ANG02] é apresentado um *middleware* baseado em agentes para suporte à colaboração entre pessoas que não necessariamente se conhecem. *Socialware* [HOY+99] provê suporte para o desenvolvimento de comunidades através da utilização de agentes pessoais e agentes de comunidades que armazenam informações compartilhadas.

As abordagens citadas anteriormente podem ser utilizadas para a construção de aplicações para o domínio de Comunidades Virtuais Móveis. Entretanto, o auxílio provido por essas abordagens é lateral, visto que as características inerentes a esse domínio não são contempladas convenientemente. Alguns trabalhos têm enfoque apenas na fase de especificação,

[1] `http://research.nokia.com/research/projects/mynet-uia/index.html`

enquanto outros consideram apenas a infra-estrutura de redes *ad-hoc* móveis na qual as aplicações são implantadas.

Em geral, fatores do domínio de Comunidades Virtuais Móveis não são tratados de forma integrada, dificultando o desenvolvimento de aplicações que possibilitem a formação de comunidades, com base na proximidade física dos dispositivos e na similaridade existente entre os usuários. Para desenvolver aplicações utilizando os trabalhos citados anteriormente, é necessário integrar as implementações de várias abordagens e uniformizar os conceitos definidos em cada uma das abordagens.

Capítulo 3

Software Baseado em Componentes

Os conceitos relacionados com software baseado em componentes, necessários para o entendimento deste trabalho, são apresentados neste capítulo. Definições sobre reutilização, flexibilidade, disponibilização e configuração de componentes, interação entre componentes e composição dinâmica são apresentadas. Além desses conceitos gerais sobre componentes, também são descritos o modelo de componentes COMPOR e o arcabouço de software no qual estão implementadas as definições desse modelo. Tanto o modelo quanto o arcabouço são utilizados na especificação e implementação dos componentes desenvolvidos neste trabalho.

3.1 Desenvolvimento Baseado em Componentes

O Desenvolvimento Baseado em Componentes (DBC) [Lau04] tem sido utilizado constantemente na produção de software, promovendo aumento de produtividade e qualidade, em virtude da reutilização de componentes previamente implementados. Um componente de software é um artefato de software que obedece a um modelo de componentes e pode ser implantado e composto independentemente, sem modificação, de acordo com um padrão de composição [HC01]. As abordagens de DBC geralmente são baseadas na arquitetura ilustrada na Figura 3.1, cujos elementos são descritos a seguir.

A *interface do componente* de software é representada pelas assinaturas dos métodos que implementam a funcionalidade do componente. O *arcabouço* de componentes é a infraestrutura que possibilita a "montagem" dos componentes para a construção de aplicações,

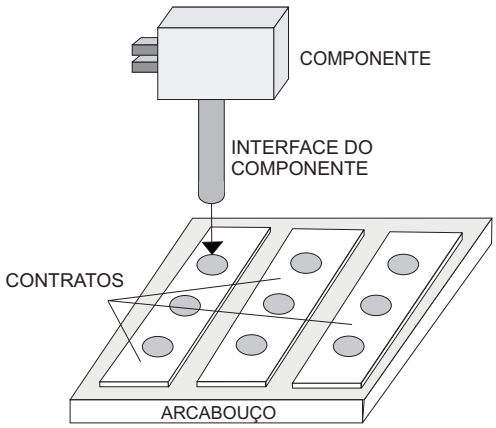

Figura 3.1: Arquitetura de componentes [FLN+04]

coordenando a interação entre os componentes. Para garantir que os componentes se comportarão da maneira esperada pelo arcabouço, são definidos os *contratos*. Os contratos representam as interfaces que o componente é obrigado a implementar. Eles garantem que o desenvolvimento de componentes independentes obedeça determinados padrões, fazendo com que o arcabouço seja capaz de utilizá-los sem conhecer detalhes internos de implementação [BBB+00].

A separação entre a interface do componente e sua implementação flexibiliza o desenvolvimento de software, uma vez que possibilita a reutilização de componentes criados por terceiros, baseando-se apenas nas suas características externas. Através dessas características, ou seja, da interface do componente, é realizado o processo de composição de componentes; a aplicação baseada em componentes é "montada" com base na ligação entre as interfaces dos componentes.

Após o processo de "montagem", os componentes são configurados de acordo com as características específicas da aplicação em questão. Nesta etapa de configuração são definidas, por exemplo, a URL de conexão de um componente de banco de dados ou a porta utilizada por um componente de comunicação. Essas informações não são conhecidas enquanto os componentes estão sendo desenvolvidos; portanto, são definidas apenas em uma etapa imediatamente anterior à execução da aplicação.

Uma vez que os componentes estão "montados", configurados e integrados ao arcabouço, a aplicação pode então ser executada. Durante a execução da aplicação, os componentes interagem uns com os outros, de quatro formas diferentes: provendo serviços para outros

componentes, acessando serviços de outros componentes, gerando eventos e observando eventos [WUK99]. Durante o ciclo de vida de execução da aplicação, os componentes podem ser substituídos por versões otimizadas ou até mesmo por versões com correção de bugs. Mediante a alteração dos requisitos da aplicação, novos componentes também podem ser adicionados, acrescentando assim novas funcionalidades ao sistema.

Para que essas alterações nos componentes que constituem a aplicação possam ser realizadas sem interferir na execução do sistema, é necessário que o arcabouço que coordena a interação entre os componentes possibilite que a aplicação seja composta dinamicamente. Ou seja, mesmo durante a execução da aplicação, novos componentes podem ser adicionados, removidos, ou alterados, permitindo que a aplicação possa evoluir de acordo com as mudanças de requisitos.

O foco na reutilização de artefatos de software previamente construídos, presente nas abordagens de DBC, é uma característica importante para o desenvolvimento de aplicações para Comunidades Virtuais Móveis. Uma vez que várias aplicações diferentes podem ser desenvolvidas para um mesmo domínio, tais como aqueles citados na Seção 2.3, é importante reutilizar artefatos de software previamente implementados. Desta forma, promove-se aumento da produtividade na concepção da aplicação, em virtude da redução do tempo necessário para o desenvolvimento obtida com a reutilização; pode-se ter também aumento da qualidade da aplicação, visto que os componentes reutilizados já podem ter sido testados e validados em outras aplicações.

3.2 Modelo de Componentes COMPOR

Como descrito na seção anterior, é necessário que haja um arcabouço para mediar a interação em aplicações cuja arquitetura é baseada em componentes de software. Geralmente esse *arcabouço de componentes* é disponibilizado para o desenvolvedor da aplicação através de um conjunto de bibliotecas de classes que são utilizadas para a implementação dos componentes.

No arcabouço de componentes são implementadas as especificações definidas em um *modelo de componentes*. Esse modelo define, em um nível conceitual, as interações que regem os componentes e as entidades de software utilizadas para a construção de aplicações. Sendo assim, antes de desenvolver uma aplicação utilizando a abordagem de componentes,

é necessário que sejam definidos o modelo e o arcabouço de componentes utilizados.

As aplicações desenvolvidas com base na infra-estrutura de software apresentada neste trabalho necessitam de suporte para a alteração em tempo de execução e para a evolução da aplicação sem uma preparação prévia (mais informações sobre essas características são descritas na Seção 3.2.2). Essas duas características estão presentes no modelo de componentes COMPOR [APF+06], desenvolvido pelo grupo de pesquisa em Composição de Software do Laboratório de Sistemas Embarcados e Computação Pervasiva. Em virtude da familiarização com a utilização deste modelo e da presença dessas duas características na sua especificação, este modelo foi escolhido para o desenvolvimento da infra-estrutura apresentada neste trabalho.

As especificações deste modelo de componentes são apresentados nesta seção, incluindo os modelos de interação baseado em serviços e eventos, propriedades de inicialização, suporte para composição dinâmica e evolução de software não antecipada. O arcabouço de componentes COMPOR é descrito na Seção 3.3.

O modelo de componentes COMPOR possui duas entidades principais: *componente funcional* e *contêiner*. Nos componentes funcionais são implementadas as funcionalidades da aplicação em questão. Os contêineres, por sua vez, não implementam funcionalidades; eles agrupam os componentes funcionais ou outros contêineres, sendo responsáveis pela mediação da interação entre os componentes funcionais. Na Figura 3.2 ilustra-se o mapeamento de um sistema baseado em módulos para o modelo de componentes COMPOR. Os módulos mais externos do sistema não implementam funcionalidades (módulo principal 1 e módulos 1.3, 1.3.3 e 1.4); eles são utilizados apenas para agrupar sub-módulos com características semelhantes. Sendo assim, esses módulos são mapeados em contêineres do modelo COMPOR. Já os outros módulos mais internos da arquitetura do sistema (módulos 1.1, 1.2, 1.3.1, 1.3.2, 1.3.3.1, 1.3.3.2, 1.4.1 e 1.4.2) implementam as funcionalidades da aplicação e, portanto, são mapeados em componentes funcionais do modelo COMPOR.

3.2.1 Modelos de Interação e Propriedades de Inicialização

O modelo de componentes COMPOR permite que os componentes funcionais interajam através de dois mecanismos: modelo de interação baseado em *serviços* e modelo de interação baseado em *eventos*. Esses dois modelos podem ser utilizados para implementar as fun-

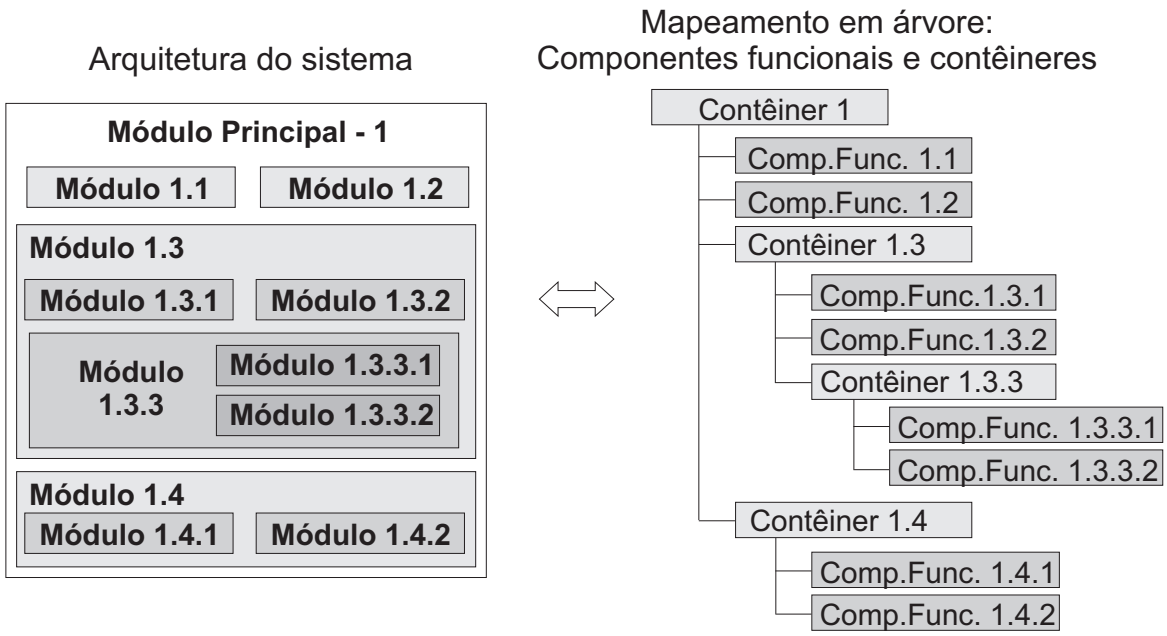

Figura 3.2: Mapeamento em árvore da arquitetura hierárquica de um sistema [Alm04]

cionalidades da aplicação nos componentes funcionais. Os componentes funcionais podem prover serviços (*serviços providos*), que podem ser acessados por quaisquer outros componentes funcionais da hierarquia que necessitem do serviço (*serviços requeridos*). A interação entre os componentes funcionais também pode ser realizada através do anúncio de eventos (*eventos anunciados*), que são recebidos por aqueles componentes funcionais da hierarquia que possuem interesse em determinado evento (*eventos de interesse*). Vale ressaltar que esses dois mecanismos de interação são implementados pelos contêineres, que implementam os mecanismos para a mediação das interações entre os componentes funcionais, que por sua vez implementam apenas as funcionalidades específicas da aplicação em questão.

Para gerenciar as interações entre os componentes, os contêineres utilizam tabelas do tipo "chave-valor": a chave utilizada é o nome do serviço e o valor é o componente que provê o serviço. O mesmo acontece com a tabela de eventos: a chave utilizada é o nome do evento e o valor são os componentes que têm interesse pelo evento. O processo de disponibilização de componentes, com a conseqüente atualização das tabelas de serviços e eventos, é apresentado na Figura 3.3 e seus passos são descritos a seguir [Alm04]. As operações de remoção e alteração de componentes são similares.

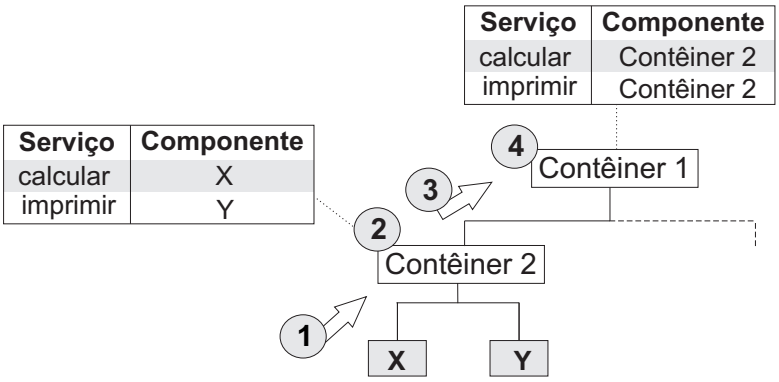

Figura 3.3: Disponibilização de componentes: atualização da tabela de serviços dos "componentes-filhos" até a raiz da hierarquia [Alm04]

1. O componente "X", que implementa o serviço "calcular", é adicionado ao Contêiner 2.

2. O Contêiner 2 atualiza a sua tabela de serviços providos pelos seus "componentes-filhos", definindo que o componente "X" implementa o serviço "calcular".

3. O Contêiner 2 solicita que o seu "contêiner-pai" (Contêiner 1) atualize a sua tabela de serviços.

4. O Contêiner 1 atualiza a sua tabela de serviços providos pelos seus "componentes-filhos", definindo que o Contêiner 2 implementa o serviço "calcular". Na verdade, quando a solicitação de atualização da tabela de serviços é feita por um contêiner, a entrada do componente adicionada na tabela é mapeada para o contêiner que realizou a solicitação de atualização (neste caso, Contêiner 2). Desta forma, a entrada na tabela indica que o Contêiner 2 "conhece" o implementador do serviço "calcular". Esta informação "conhece o implementador" está definida na tabela de serviços do Contêiner 2, apontando para o real implementador, o componente "X").

Após esse processo, os serviços do componente "X" podem ser acessados a partir de qualquer componente da hierarquia. Os mesmos passos descritos anteriormente são seguidos para a atualização da tabela de eventos de interesse. Da mesma forma, após a execução desses passos, o componente "X" poderá receber os anúncios de todos os seus eventos de interesse.

Além dos modelos de interação baseado em serviços e baseado em eventos, os desenvolvedores de componentes funcionais do modelo COMPOR podem utilizar as *propriedades de inicialização*. Através dessas propriedades, é possível postergar a definição de configurações do componente que não são conhecidas *a priori* — em tempo compilação; por exemplo, a porta utilizada por um componente de comunicação ou a URL utilizada por um componente de banco de dados. Ao utilizar o componente, o "montador" da aplicação define o valor das propriedades. Essas propriedades são repassadas para o componente durante a sua inicialização, permitindo com isso que o componente possa utilizar os dados configurados pelo "montador" da aplicação.

3.2.2 Composição Dinâmica e Evolução de Software não Antecipada

De acordo com as duas hierarquias baseadas no modelo de componentes COMPOR, ilustradas nas Figuras 3.2 e 3.3, é possível identificar que não existe dependência entre componentes funcionais. Um componente funcional possui referência apenas para o seu "contêiner-pai". Essa característica peculiar do modelo de componentes COMPOR facilita a implementação de *composição dinâmica* nos arcabouços que seguem as especificações do modelo. Uma vez que não existe referência entre os componentes funcionais e que a interação entre os componentes é realizada através dos contêineres, a tarefa de trocar componentes, inclusive em tempo de execução, é simplificada. Para isso, é necessário apenas adicionar o novo componente ao contêiner; de acordo com o processo de disponibilização apresentado anteriormente, o valor do novo componente será atualizado na tabela de serviços/eventos e as novas requisições serão direcionadas automaticamente para o novo componente adicionado.

Outra característica presente no modelo de componentes COMPOR é o suporte para *evolução de software não antecipada*. Ao desenvolver aplicações com base no modelo COMPOR, não é necessário se preparar para as possíveis mudanças que ocorrerão no software, ou seja, não é necessário antecipar as mudanças. A possibilidade de adicionar, remover e alterar componentes, inclusive em tempo de execução, permite que o software possa evoluir sem que haja uma preparação prévia para isso.

Essas duas características presentes no modelo de componentes COMPOR — composição dinâmica e evolução de software não antecipada — são também importantes para o desenvolvimento de aplicações para Comunidades Virtuais Móveis.

Os dispositivos nos quais esse tipo de aplicação é implantando possuem restrições de processamento e memória. Isso dificulta, por exemplo, a disponibilização prévia de componentes que permitam a utilização de várias tecnologias de comunicação sem fio — normalmente presentes nos dispositivos móveis. O suporte para composição dinâmica possibilita que mediante a necessidade de comunicação através de Bluetooth, por exemplo ao se aproximar de outro dispositivo com esta tecnologia, um novo componente Bluetooth possa ser adicionado à aplicação, de forma transparente, sem que haja a necessidade do usuário interromper os serviços que já estejam sendo utilizados.

Por sua vez, evolução de software não antecipada é importante para as aplicações para Comunidades Virtuais Móveis em virtude da constante mudança presente nos domínios nos quais esse tipo de aplicação é utilizado. Por exemplo, um algoritmo mais eficiente para a descoberta de dispositivos é implementado e, portanto, é interessante que seja acrescentado à aplicação com facilidade; um novo padrão de interoperabilidade entre aplicações é definido e as aplicações já desenvolvidas devem estar em conformidade com este padrão para que possam interagir com novas aplicações. É importante então que o modelo dê suporte para a alteração das funcionalidades que certamente serão realizadas na aplicação, sem demandar nenhum esforço prévio adicional por parte dos desenvolvedores.

3.3 Arcabouço de Componentes COMPOR

O modelo de componentes COMPOR, apresentado na seção anterior, permite a concepção de sistemas apenas até a fase de projeto, assim como qualquer outro modelo de componentes, visto que o modelo define apenas uma especificação. Para implementar e executar um sistema projetado com base em tal modelo, é necessário um arcabouço de software que implemente as especificações e conceitos definidos pelo modelo de componentes. No caso do modelo COMPOR, este arcabouço deve permitir a criação de componentes funcionais e contêineres, a criação da hierarquia do sistema com a disponibilização dos componentes, a criação de serviços e eventos, a execução do sistema, etc.

Na Figura 3.4 é ilustrado o diagrama das principais classes de um arcabouço de software que implementa as especificações do modelo de componentes COMPOR. Este diagrama representa a modelagem de um arcabouço implementado na linguagem Java, denominado JCF

(*Java Component Framework*) [APPC04]; a notação utilizada no diagrama é especificada pela linguagem UML (*Unified Modeling Language*) [BRJ00].

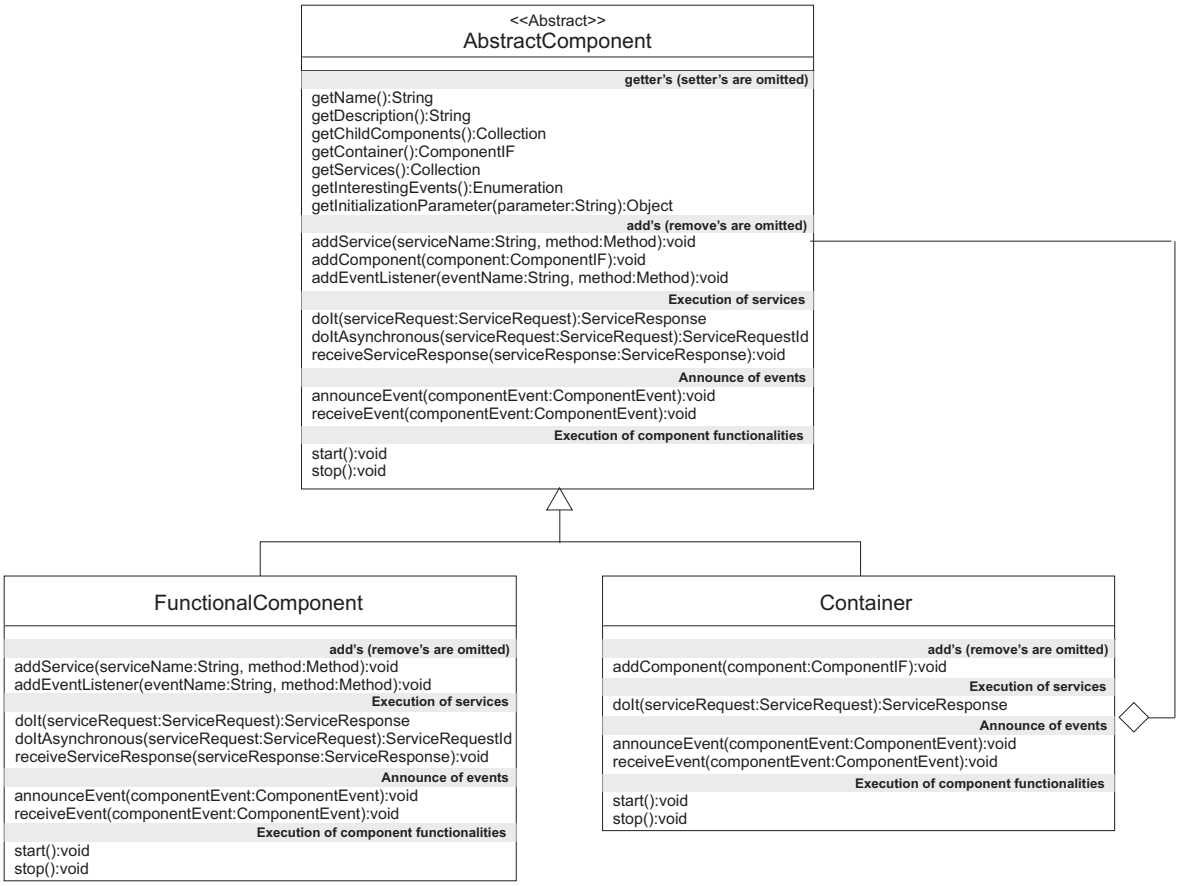

Figura 3.4: Diagrama das classes principais do arcabouço de software JCF

As instâncias da classe *FunctionalComponent* representam os componentes funcionais definido no modelo e as instâncias da classe *Container* representam os contêineres. Essas duas classes herdam a implementação da classe *AbstractComponent*, na qual são implementados os métodos comuns aos componentes funcionais e aos contêineres, tais como os métodos *get* e *set*. Além disso, na classe *AbstractComponent* também são definidas as interfaces dos métodos que devem ser implementados pelos componentes funcionais e pelos contêineres, tais como os métodos de adição e remoção de componentes, serviços e eventos; execução de serviços; anúncio de eventos; e inicialização e finalização dos componentes. Em cada uma das subclasses, *FunctionalComponent* e *Container*, esses métodos são implementados de maneira diferente, de acordo com as especificações definidas no modelo de componentes COMPOR.

A agregação entre *Container* e *AbstractComponent*, ilustrada no diagrama, é baseada no padrão de projeto *Composite* [GHJV95]. Este padrão é utilizado para modelar estruturas hierárquicas baseadas em composição recursiva. Neste caso, a agregação permite que *Container*s possam conter *FunctionalComponent*s e/ou outros *Container*s, uma vez que os objetos dessas classes também são do tipo *AbstractComponent*. Além disso, a utilização desse padrão facilita a implementação do arcabouço, pois permite aos clientes (*Container*s) tratarem de maneira uniforme objetos individuais (*FunctionalComponent*s) e composição de objetos (*Container*s) [GHJV95].

3.3.1 Execução de Aplicações com Suporte para Composição Dinâmica

Para que as aplicações desenvolvidas com base no modelo de componentes COMPOR possam ser executadas com suporte para composição dinâmica, é necessário que a linguagem na qual o arcabouço seja implementado possibilite o *carregamento de classes* em tempo de execução e a *reflexão* sobre a estrutura das classes.

O arcabouço JCF é implementado na linguagem Java e, portanto, pode utilizar o mecanismo de *class loaders* [Sun95] disponível na linguagem Java. Através desse mecanismo é possível adicionar classes de componentes que não foram "carregados" inicialmente com a aplicação. Novas classes podem ser acrescentadas ao *classpath* da aplicação durante a sua execução, sem a necessidade de interromper a execução do sistema. Com isso, novos componentes podem ser adicionados, alterados ou removidos da aplicação.

Porém, o mecanismo de carregamento de classes não é capaz de identificar/invocar os serviços/eventos disponibilizados pelos componentes adicionados à aplicação. Para isso, é necessário utilizar o mecanismo de reflexão. A API[1] *Reflection* [Sun95] da linguagem Java permite recuperar e invocar os métodos que representam os serviços e eventos dos componentes. Com isso, é possível invocar serviços e anunciar eventos que não eram "conhecidos" ao iniciar a aplicação.

[1] *Application Programming Interface*

3.3.2 Implementação em Várias Plataformas

Atualmente existem implementações do arcabouço COMPOR em duas plataformas de desenvolvimento, além de uma implementação em andamento para a linguagem C++. Essas duas implementações existentes foram desenvolvidas para a tecnologia Java [Fla05], que é dividida em três plataformas: Java SE (*Java 2 Platform, Standard Edition*), utilizada no desenvolvimento de aplicações desktop; Java EE (*Java 2 Platform, Enterprise Edition*), direcionada para aplicações corporativas; e Java ME (*Java 2 Platform, Micro Edition*), direcionada para dispositivos com recursos de processamento e memória restritos.

A primeira versão do arcabouço — *Java Component Framework*, apresentado na Seção 3.3 — foi implementada para a plataforma Java SE, contemplando todas as especificações definidas no modelo de componentes COMPOR, inclusive a execução de aplicações com suporte para composição dinâmica. Outras duas versões do arcabouço foram implementadas para a plataforma Java ME: uma para dispositivos com recursos bastante restritos e outra para dispositivos com um maior poder computacional, quando comparados com os primeiros. A seguir, a arquitetura da plataforma Java ME é detalhada e são apresentadas algumas características das versões do arcabouço para esta plataforma. Um foco maior é dado à plataforma Java ME visto que os componentes de software desenvolvidos neste trabalho são direcionados para dispositivos nos quais essa plataforma é implantada.

A plataforma Java ME [Yua03] é direcionada para dispositivos com recursos de processamento e memória restritos, tais como telefones celulares, PDAs e *set-top boxes*. A arquitetura da plataforma Java ME, ilustrada na Figura 3.5, é dividida em três níveis: *Configuration*, *Profile* e *Optional Package*. *Configuration* é um conjunto básico de APIs e características de máquina virtual necessárias para dar suporte a um amplo conjunto de dispositivos. *Profile* é um conjunto adicional de APIs que dão suporte a um conjunto mais restrito de dispositivos — um *Profile* possui dependência direta de uma *Configuration* específica. Por fim, *Optional Packages* provêem APIs para tecnologias específicas. Com base nesses três níveis, para que se possa configurar um ambiente de execução para aplicações Java ME, é preciso necessariamente ter uma *Configuration* e um *Profile*, além de qualquer número de *Optional Packages*. Por exemplo, um ambiente de execução Java ME válido pode ser formado pela *Configuration* CDC, pelo *Profile* PP e pelos *Optional Packages* RMI e JDBC [Sun05].

Como pode ser observado na Figura 3.5, existem duas *Configurations* na arquitetura da

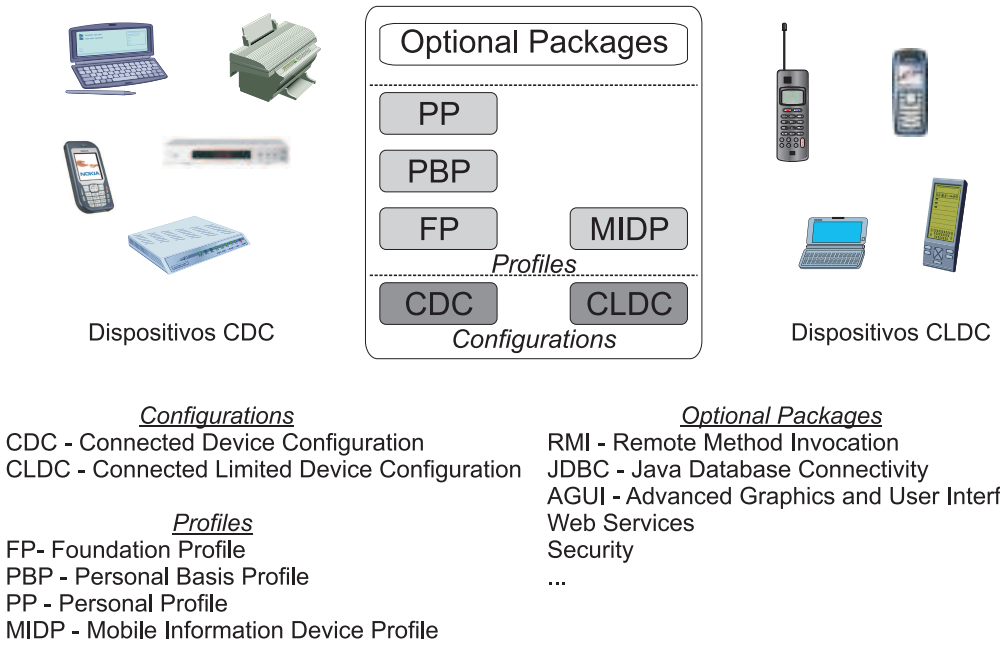

Figura 3.5: Arquitetura da plataforma Java ME

plataforma Java ME: CLDC e CDC; para cada uma delas existe uma implementação do arcabouço de componentes COMPOR. CLDC foi projetada tendo em vista os requisitos restritos de memória dos telefones celulares [Sun05]. Para atender a esses requisitos foram realizadas algumas alterações na máquina virtual Java, tal como a eliminação de ponto flutuante; e outras alterações na biblioteca de classes, tais como a remoção de reflexão e de *class loaders* específicos para uma determinada aplicação. Essas limitações encontradas na *Configuration* CLDC restringiram a implementação do suporte para composição dinâmica no arcabouço COMPOR para CLDC. Entretanto, o restante da especificação do modelo está implementado na versão do arcabouço para CLDC: modelo de interação baseado em serviços, modelo de interação baseado em eventos, propriedades de inicialização e suporte para evolução de software não antecipada.

Por outro lado, CDC foi projetado com o objetivo de ser compatível com Java SE e dar suporte a dispositivos com recursos restritos [Sun05]. CDC é totalmente compatível com a especificação da máquina virtual Java. Sua biblioteca de classes possui pequenas alterações com relação à biblioteca de Java SE, objetivando a conservação de recursos. Com isso, os desenvolvedores podem tirar proveito das ferramentas e bibliotecas já utilizadas em aplicações Java SE, além das características de um ambiente de execução já consolidado.

Sendo assim, a implementação do arcabouço COMPOR para CDC é compatível com as especificações definidas pelo modelo — inclusive o suporte para composição dinâmica — visto que esta plataforma é semelhante à plataforma Java SE.

3.3.3 Utilização em Aplicações para Comunidades Virtuais Móveis

As aplicações para Comunidades Virtuais Móveis devem ser executadas em dispositivos que facilitem a mobilidade e a conectividade dos seus usuários. Desta forma, dentre as duas plataformas de desenvolvimento com versões implementadas do arcabouço COMPOR (Java SE e Java ME), a plataforma Java ME é mais adequada para o desenvolvimento dessas aplicações, uma vez que seus dispositivos são mais apropriados para esses domínios.

Dada a importância do suporte para composição dinâmica e da evolução de software não antecipada nas aplicações para Comunidades Virtuais Móveis (discutida na Seção 3.2.2), a versão CDC é mais adequada dentre as duas versões do arcabouço para a plataforma Java ME (CLDC e CDC), uma vez que contempla essas características.

Sendo assim, os componentes desenvolvidos neste trabalho utilizam a versão do arcabouço COMPOR para a plataforma Java ME, mais especificamente para a *Configuration* CDC. Tais componentes são apresentados no próximo capítulo.

3.4 Sumário do Capítulo

Neste capítulo foram apresentados os conceitos da abordagem de desenvolvimento baseado em componentes relevantes para o entendimento deste trabalho. Em seguida foram apresentadas características do modelo de componentes COMPOR, tais como os modelos de interação (serviços e eventos) e as propriedades de inicialização. Foram discutidas também duas características presentes neste modelo: suporte para a composição de dinâmica de aplicações e suporte para a evolução de software não antecipada. Por fim, foi apresentado o arcabouço de componentes COMPOR, discutindo as características inerentes a sua implementação em várias plataformas, assim como a sua utilização em aplicações para Comunidades Virtuais Móveis.

Capítulo 4

Infra-estrutura de Componentes para Comunidades Virtuais Móveis

Neste capítulo detalha-se a infra-estrutura de software baseada em componentes para facilitar a construção de aplicações para Comunidades Virtuais Móveis. Esta infra-estrutura contempla um conjunto de requisitos definidos com base na identificação de características presentes nos domínios de aplicação nos quais os conceitos de Comunidades Virtuais Móveis são aplicados. Para a definição dos requisitos, apresentada na Seção 4.1, também levou-se em consideração a análise dos trabalhos relevantes ao tema discutidos na Seção 2.4.

Na Seção 4.2 é apresentada uma breve descrição dos componentes definidos para a infra-estrutura, assim como as relações de dependência existentes entre eles. A especificação mais detalhada destes componentes é apresentada na Seção 4.3, assim como a definição dos serviços providos e requeridos por cada componente, os eventos anunciados e de interesse de cada componente, além das propriedades de inicialização. Nesta seção também são detalhadas as interações existentes entre os componentes da infra-estrutura.

Para cada um dos componentes foi realizada uma implementação, cujos detalhes são apresentados na Seção 4.4. Estas implementações são utilizadas como base para o estudo de caso apresentado no Capítulo 5. Por fim, na Seção 4.5 é apresentado um sumário do que é apresentado neste capítulo.

4.1 Requisitos de Aplicações para Comunidades Virtuais Móveis

Os *usuários* de aplicações para Comunidades Virtuais Móveis são indivíduos que utilizam *dispositivos* móveis interconectados, constituindo *comunidades* de acordo com *interesses* comuns. Estes indivíduos são notificados da *proximidade* de outros indivíduos e de acordo com a *similaridade* entre os interesses dos mesmos, novas comunidades podem ser formadas. Nestas comunidades são compartilhados *conteúdos* que são acessados de acordo com os níveis de *autorização* de cada indivíduo.

Com base na descrição anterior sobre as características e entidades presentes em aplicações de Comunidades Virtuais Móveis, foram especificados os requisitos necessários para uma infra-estrutura de software cujo propósito central é facilitar a construção de tais aplicações. Estes requisitos são descritos a seguir.

- **Requisito 1 - Notificação da proximidade de indivíduos -** A notificação da proximidade é importante para promover o crescimento da rede de relacionamentos dos indivíduos e a interação entre os mesmos. Quando um indivíduo recebe a notificação da aproximação de outro indivíduo que já faz parte da sua lista de contatos, ele pode a partir de então estabelecer comunicação com o indivíduo encontrado. Caso receba a notificação da aproximação de um indivíduo desconhecido, porém com interesses similares, ele pode adicionar o indivíduo encontrado à sua lista de contatos, aumentando desta forma sua rede de relacionamentos.

- **Requisito 2 - Notificação da proximidade de dispositivos -** Para que haja a notificação da proximidade de indivíduos é necessário que anteriormente ocorra a notificação da aproximação dos dispositivos utilizados pelos indivíduos. A partir do momento em que ocorre a notificação da proximidade de um dispositivo, é possível notificar a aproximação do indivíduo que utiliza o dispositivo encontrado.

- **Requisito 3 - Formação de comunidades -** Deve ser possível para indivíduos com interesses similares constituir comunidades nas quais informações de interesse comum possam ser compartilhadas. Estas comunidades são formadas de acordo com a proximidade física dos indivíduos e, portanto, devem fazer sentido somente na área de

cobertura da conexão existente entre os dispositivos.

- **Requisito 4 - Identificação da similaridade entre indivíduos -** É importante que seja possível identificar a similaridade existente entre os indivíduos. Desta forma, os indivíduos somente serão notificados da proximidade de indivíduos relevantes, ou seja, que possuem interesses similares aos seus. Esta medida de similaridade atua como um filtro que evita o excesso de notificações em locais com grande concentração de pessoas, tais como estádios de futebol, *shows* e *shopping centers*. Caso não exista um filtro baseado na similaridade, os indivíduos serão constantemente importunados com a notificação de indivíduos com os quais não possuem semelhança.

- **Requisito 5 - Representação dos interesses de indivíduos -** É necessário que os interesses dos indivíduos sejam representados nas aplicações de Comunidade Virtuais Móveis para que se possa realizar a comparação das características dos indivíduos e com isso identificar a similaridade existente entre eles.

- **Requisito 6 - Disponibilização de conteúdo em comunidades -** Os indivíduos que fazem parte de uma comunidade devem poder disponibilizar conteúdo, tais como documentos, planilhas eletrônicas, artigos, apresentações, etc. Este conteúdo pode ser compartilhado com outros indivíduos que também fazem parte da comunidade, promovendo desta forma a disseminação de informação de interesse comum.

- **Requisito 7 - Restrição de acesso ao conteúdo das comunidades -** É importante definir restrições de acesso ao conteúdo disponibilizado nas comunidades de forma que seja possível atribuir papéis aos indivíduos e filtrar o acesso ao conteúdo com base nos papéis que os indivíduos desempenham. Indivíduos não autorizados não devem ter acesso a conteúdos que não estão disponíveis para os papéis que desempenham.

4.2 Componentes para Comunidades Virtuais Móveis

A infra-estrutura apresentada neste trabalho utiliza componentes de software como entidade principal para realizar a composição de aplicações. Deste modo, os requisitos descritos na seção anterior foram agrupados em componentes, de forma que o conjunto de requisitos seja

contemplado pela união das especificações dos componentes. Os requisitos foram agrupados em cinco componentes, de acordo com a organização apresentada na Tabela 4.1.

—	**Componentes**				
Requisitos	UserSearcher	UserManager	DeviceSearcher	Similarity	ContentSharing
Requisito 1	✓				
Requisito 2			✓		
Requisito 3		✓			
Requisito 4				✓	
Requisito 5		✓			
Requisito 6					✓
Requisito 7					✓

Tabela 4.1: Organização dos requisitos em componentes

O componente `UserSearcher` é responsável pela busca de usuários próximos a um determinado usuário, portanto contempla a *notificação da proximidade de indivíduos* ①. A *formação de comunidades* ③ e a *representação dos interesses de indivíduos* ⑤ estão relacionadas às informações inerentes ao usuário; desta forma estão agrupadas no componente `UserManager`, responsável pelo gerenciamento do acesso a estas informações. A busca por dispositivos próximos ao dispositivo do usuário é realizada pelo componente `DeviceSearcher` que conseqüentemente contempla a *notificação da proximidade de dispositivos* ②. A *identificação da similaridade entre indivíduos* ④ é contemplada pelo componente `Similarity`. Por fim, o componente `ContentSharing` implementa os mecanismos de *disponibilização de conteúdo em comunidades* ⑥, assim como a *restrição de acesso a este conteúdo disponibilizado* ⑦.

Na Figura 4.1 são ilustrados os componentes que fazem parte da infra-estrutura de software e as dependências existentes entre eles. A seguir, são descritas as dependências existentes entre cada um dos componentes. Mais detalhes sobre a especificação dos componentes são documentados na Seção 4.3.

- **Dependência entre os componentes `UserSearcher` e `Similarity` -** O componente `UserSearcher` utiliza o componente `Similarity` para realizar o cálculo da similaridade entre os indivíduos.

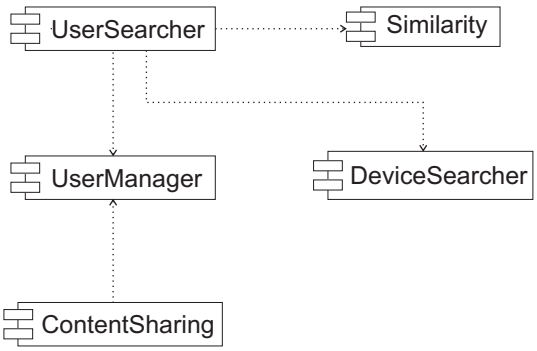

Figura 4.1: Componentes para Comunidades Virtuais Móveis

- **Dependência entre os componentes `UserSearcher` e `DeviceSearcher` -** Para que o componente `UserSearcher` possa notificar a aproximação de indivíduos é necessário que ele possa realizar buscas pelos dispositivos utilizados pelos indivíduos. A inicialização e interrupção da busca por dispositivos, assim como a notificação da proximidade de dispositivos são realizadas pelo componente `DeviceSearcher`.

- **Dependência entre os componentes `UserSearcher` e `UserManager` -** O componente `UserSearcher` recupera os interesses do usuário a partir do componente `UserManager`; estes interesses são utilizados para avaliar a similaridade dos indivíduos através do componente `Similarity`. Além disto, o componente `UserSearcher` depende do componente `UserManager` para recuperar informações relativas à lista de contatos do usuário.

- **Dependência entre os componentes `ContentSharing` e `UserManager` -** O componente `ContentSharing` recupera a lista de comunidades das quais o usuário faz parte através do componente `UserManager`. A lista de comunidades é utilizada na disponibilização e na restrição de acesso ao conteúdo da comunidade.

O agrupamento dos requisitos em componentes foi realizado para obter um menor nível de acoplamento entre os componentes, uma maior coesão das funcionalidades e conseqüentemente um maior nível de reutilização destes componentes. Com base nesses princípios, o componente `UserManager` implementa o gerenciamento de todas as informações relativas ao usuário, o componente `UserSearcher` é responsável por todas as funcionalidades inerentes à busca de usuários e o componente `DeviceSearcher` encapsula todos os algoritmos relacionados à busca de dispositivos. O componente `ContentSharing` implementa

o gerenciamento de todo o acesso ao conteúdo das comunidades e, por fim, o componente `Similarity` identifica somente a similaridade entre usuários.

4.3 Especificação dos Componentes

Nesta seção é apresentada a especificação dos componentes descritos na seção anterior. Esta especificação tem como base o modelo de componentes COMPOR e, portanto, utiliza os conceitos de componentes funcionais, modelo de interação baseado em serviços (serviços providos e serviços requeridos), modelo de interação baseado em eventos (eventos anunciados e eventos de interesse) e propriedades de inicialização.

Na Figura 4.2 são apresentadas as propriedades de inicialização, os serviços e os eventos que fazem parte da especificação dos componentes da infra-estrutura de software. São também ilustradas as dependências existentes entre os componentes. O detalhamento completo da especificação é apresentado no Apêndice A.

A interação entre os componentes da infra-estrutura durante a execução de uma aplicação para Comunidades Virtuais Móveis envolve a invocação de serviços e o disparo de eventos. A seguir é descrita uma seqüencia de execução de uma aplicação que utiliza a infra-estrutura apresentada neste trabalho, descrevendo os serviços invocados e os eventos anunciados.

Busca de dispositivos/usuários e cálculo de similaridade

Uma aplicação de Comunidade Virtuais Móveis é inicializada a partir da invocação do serviço `searchNearbyUsers`, provido pelo componente `UserSearcher` (este serviço retorna o identificador da busca para que o cliente do componente possa interromper a busca posteriormente, através do serviço `stopUserSearcher`). Na implementação do serviço `searchNearbyUsers`, é requisitada a busca por dispositivos móveis próximos ao dispositivo do usuário que está utilizando a aplicação. Esta busca por dispositivos é implementada no serviço `searchNearbyDevices`, provido pelo componente `DeviceSearcher` (de forma análoga à busca por usuários, este serviço também retorna o identificador da busca para que o cliente do componente possa interromper a busca posteriormente, através do serviço `stopDeviceSearcher`). Ao localizar novos dispositivos, o serviço `searchNearbyDevices` anuncia o evento `deviceFound`, informando o identificador

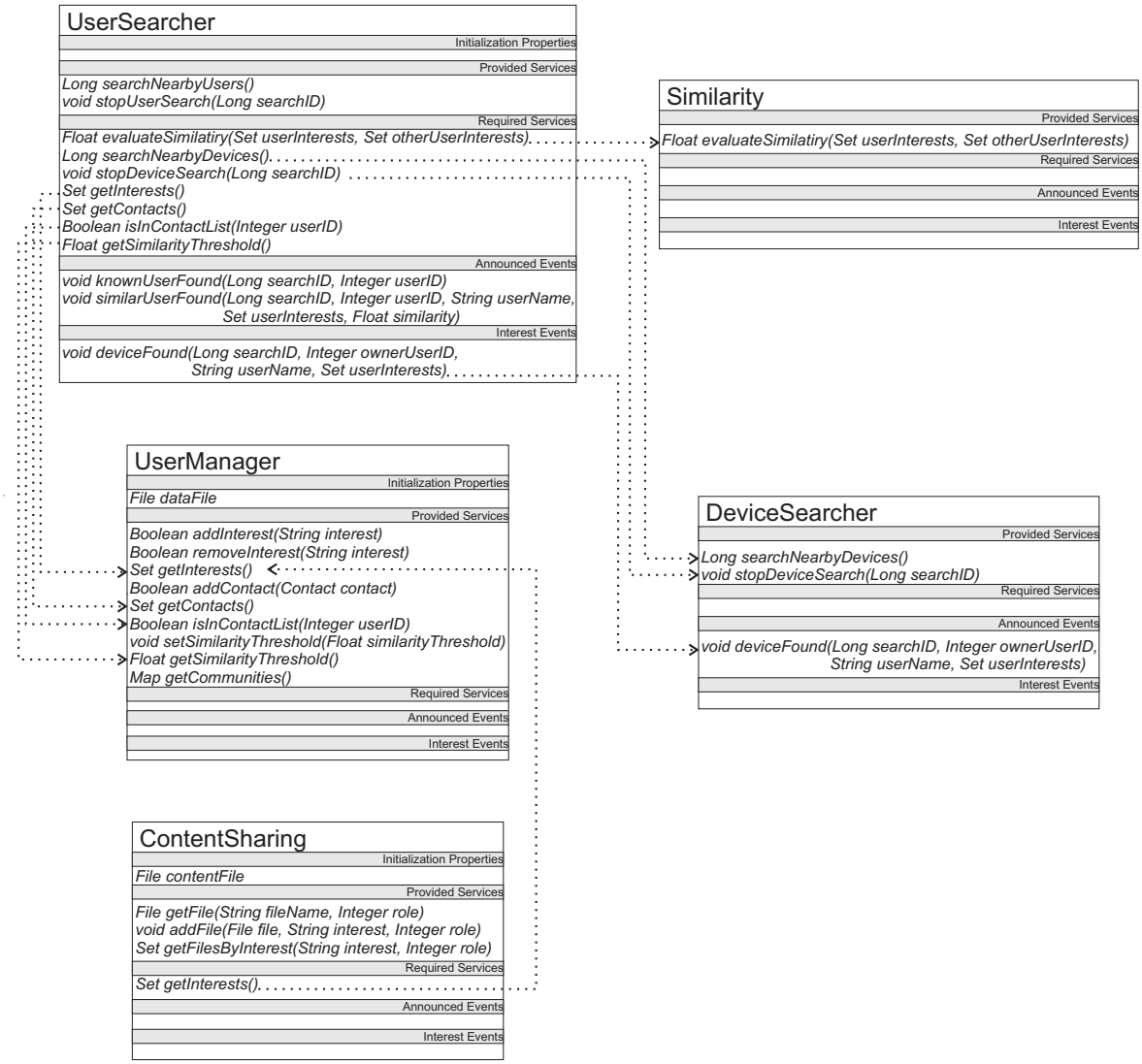

Figura 4.2: Especificação de componentes para Comunidades Virtuais Móveis

do usuário que utiliza o dispositivo localizado.

Ao receber notificações sobre os dispositivos encontrados (através do evento de interesse `deviceFound`), o componente `UserSearcher` realiza algumas verificações com base no identificador do usuário encontrado. Caso o usuário que utiliza o dispositivo esteja na lista de contatos (recuperada através do serviço `getContacts` do componente `UserManager`), o componente anuncia o evento `knownUserFound` para notificar que um usuário já conhecido foi encontrado. Caso contrário, o componente recupera os interesses do usuário (através do serviço `getInterests` do componente `UserManager`) e requisita o cálculo da avaliação de similaridade entre os usuários através do serviço `evaluateSimilarity` do componente `Similarity`.

Se o valor da similaridade estiver acima do limiar de similaridade (recuperado através do serviço `getSimilarityThreshold` do componente `UserManager`), o evento `similarUserFound` é anunciado para notificar que um usuário com características similares foi encontrado. Caso o valor da similaridade esteja abaixo do limiar de similaridade, nenhuma notificação é realizada, uma vez que o usuário não deve ser notificado da proximidade de outros usuários com os quais não possui características similares.

Disponibilização de conteúdo e acesso às informações do usuário

O conteúdo compartilhado nas comunidades é disponibilizado através do serviço `getFile` do componente `ContentSharing`; a lista de arquivos associados a uma determinada comunidade é recuperada através do serviço `getFilesByInterest`. Ao adicionar conteúdo a uma determinada comunidade, através do serviço `addFile`, o componente `ContentSharing` verifica se o arquivo compartilhado está associado a um interesse que realmente faz parte da lista de interesses do usuário (através do serviço `getInterests` do componente `UserManager`).

O componente `UserManager` ainda disponibiliza serviços para a adição/remoção de interesses do usuário (`addInterest` e `removeInterest`), a definição do limiar de similaridade do usuário (`setSimilarityThreshold`), a recuperação da lista de comunidades do usuário (`getCommunities`) e o acesso a lista de contatos do usuário (`addContact` e `isInContactList`).

4.4 Implementação dos Componentes

Na Seção 4.1 foram apresentados os requisitos identificados em aplicações para Comunidades Virtuais Móveis. Esses requisitos foram agrupados em componentes que constituem a especificação da infra-estrutura de software apresentada na Seção 4.3. Nessa especificação são descritos os componentes e as interações existentes entre eles, de acordo com as suas dependências.

Para avaliar e validar a especificação da infra-estrutura apresentada anteriormente, foi implementada uma versão funcional de cada um dos componentes, para a plataforma Java ME CDC. Estas implementações dos componentes servem como base para a implementação

do estudo de caso apresentado no Capítulo 5. Ao longo desta seção são discutidas informações inerentes à implementação dos componentes da infra-estrutura, tais como algoritmos e estruturas de dados utilizados.

Componente `UserManager`

Este componente gerencia o acesso às informações do usuário, tais como nome, limiar de similaridade, lista de contatos, interesses e comunidades. Entre duas utilizações simultâneas deste componente, as informações do usuário precisam ser armazenadas para que posteriormente possam ser recuperadas, evitando que o usuário forneça suas informações mais de uma vez. A solução de persistência das informações utilizada nesta implementação do componente é o armazenamento em um arquivo de propriedades com pares chave/valor. A definição e recuperação destas propriedades são realizadas através da classe `java.util.Properties`[1] da API de Java. A seguir é apresentado um trecho deste arquivo de propriedades.

```
interests = leitura, futebol, música
similarityThreshold = 0.85
```

As informações do usuário são definidas/recuperadas através das chaves (*interests* e *similarityThreshold*). O caminho do arquivo é definido através da propriedade de inicialização `dataFile` definida pelo componente.

Durante a inicialização do componente as informações armazenadas no arquivo de propriedades são carregadas para a memória. Caso o arquivo ainda não exista (na primeira utilização do componente na aplicação), ele é criado com valores vazios para as propriedades. Após a inicialização do componente, as informações do usuário são recuperadas da memória. As alterações nas informações do usuários (através dos serviços *add*, *remove* e *set*) são realizadas na memória e também propagadas para o arquivo de propriedades.

O nome, limiar de similaridade, lista de contatos e interesses do usuário são armazenados diretamente no arquivo de propriedades. Entretanto, a lista de comunidades das quais o usuário participa não são armazenadas, uma vez que podem ser recuperadas através da lista

[1] `http://java.sun.com/docs/books/tutorial/essential/environment/properties.html`

de contatos e de interesses. O Algoritmo 4.1 é utilizado na recuperação das comunidades das quais o usuário participa.

Algoritmo 4.1 Recuperação de comunidades de um usuário

```
conjuntoContatosUsuario ← conjunto de contatos do usuário
conjuntoInteressesUsuario ← conjunto de interesses do usuário
comunidadesUsuario ← comunidades do usuário
para cada contato ∈ conjuntoContatosUsuario faça
  conjuntoInteressesContato ← conjunto de interesses do contato
  para cada interesseContato ∈ conjuntoInteressesContato faça
    se interesseContato ∈ conjuntoInteressesUsuario então
      comunidade ← cria comunidade com o nome do interesseContato
      associa contato a comunidade
      atualiza comunidadesUsuario com os dados da comunidade
    fim se
  fim para
fim para
return comunidadesUsuario
```

A implementação deste algoritmo na linguagem Java utiliza o tipo `java.util.Set` para representar os conjuntos de contatos e interesses. A associação de contatos para comunidades utiliza o tipo `java.util.Map`.

Componente `UserSearcher`

O algoritmo de busca por usuários neste componente é iniciado com a invocação do serviço `searchNearbyUsers`. Este serviço, por sua vez, invoca o serviço de busca de dispositivos `searchNearbyDevices`, provido pelo componente `DeviceSearcher`, e retorna um identificador da busca para o cliente do componente. Este identificador é utilizado pelo cliente para interromper a busca por usuários através do serviço `stopUserSearcher`. Neste serviço, por sua vez, é solicitada a interrupção da busca por dispositivos, através da invocação do serviço `stopDeviceSearch`.

A implementação da busca por usuários utiliza o modelo de interação baseado em

eventos provido pelo modelo de componentes COMPOR. Sempre que o componente `DeviceSearcher` encontra um dispositivo, o evento `deviceFound` é anunciado com as informações do dispositivo. Uma vez que o componente `UserSearcher` possui o evento `deviceFound` como um dos seus eventos de interesse, ele recebe as notificações. O Algoritmo 4.2 é implementado no método que recebe as notificações do evento `deviceFound` no componente `UserSearcher`.

Algoritmo 4.2 Recebimento das notificações do evento `deviceFound`

```
estaNaListaDeContatos ← retorno do serviço isInContactList
se estanaListaDeContatos então
    anuncia o evento knownUserFound
senão
    similaridadeAvaliada ← retorno do serviço evaluateSimilarity
    limiarSimilaridade ← retorno do serviço getSimilarityThreshold
    se similaridadeAvaliada ≥ limiarSimilaridade então
        anuncia o evento similarUserFound
    fim se
fim se
```

Em resumo, o componente `UserSearcher` faz um mapeamento dos eventos relacionados à descoberta de dispositivos (`deviceFound`) para eventos relacionados à descoberta de usuários (`knownUserFound` e `similarUserFound`). O anúncio destes dois últimos eventos é realizado com base nas informações recebidas através do evento `deviceFound` (identificador do usuário que utiliza o dispositivo e o seu conjunto de interesses).

Componente `DeviceSearcher`

A implementação da busca por dispositivos presente neste componente utiliza o mecanismo de descoberta de nós disponibilizado pelo *middleware Wings* [Lou06]. O propósito deste *middleware* é a disponibilização de serviços em ambientes pervasivos — ambientes nos quais a computação está embutida nos objetos do dia-a-dia, tais como televisores, carros e roupas; nestes ambientes, os objetos se comunicam de forma transparente para nos apresentar informações e recursos independente da hora e do lugar, de acordo com as nossas necessidades e

preferências.

A arquitetura do *Wings* é composta por quatro módulos, de acordo com a ilustração apresentada na Figura 4.3. O módulo de *Evolução Dinâmica* provê mecanismos para possibilitar que o *middleware* possa ser atualizado tem tempo de execução. O módulo de *Ciência de Contexto* disponibiliza informações sobre o contexto do usuário, do ambiente e do dispositivo, para as aplicações. No módulo de *Redes Pervasivas* estão implementadas mecanismos para disponibilização de serviços e para descoberta de nós. Por fim, o módulo de *Fachada* provê um ponto único de acesso às funcionalidades do *middleware Wings*.

Figura 4.3: Arquitetura do *middleware Wings* [Lou06]

Para implementar as funcionalidades providas em cada um dos módulos citados anteriormente, é necessário desenvolver *Plug-ins de Ciência de Contexto*, *Plug-ins de Disponibilização de Serviços* e *Plug-ins de Descoberta de Nós*. Na implementação deste componente são utilizados os *Plug-ins de Descoberta de Nós*. Estes *plug-ins* permitem que sejam utilizadas várias tecnologias para a descoberta de dispositivos, tais como Bluetooth e UPnP[2], proporcionando independência da tecnologia de rede utilizada. A seguir são apresentados os passos utilizados na integração do componente `DeviceSearcher` com o mecanismo de descoberta de nós disponibilizado pelo *Wings*.

1. **Adicionar *Plug-ins de Descoberta de Nós* -** Os algoritmos para descoberta de nós são implementados pelos *plug-ins*. Sendo assim, ao menos um *plug-in* deve ser adicionado ao *middleware* para que a descoberta seja realizada. O componente `DeviceSearcher` utiliza o *Bluetooth Host Discovery Plug-in*, provido pelo Wings; este *plug-in* utiliza os protocolos de descoberta de nós da tecnologia Bluetooth. O *plug-in* é adicionado durante a instanciação do componente, de acordo com o Código 4.1.

[2]Universal Plug and Play `http://www.upnp.org`

Código 4.1: Adição de *Plug-in de Descoberta de Nós*

```
public class DeviceSearcherComponent
  extends FunctionalComponent
  implements HostDiscoveryListener {
  ...
  public DeviceSearcherComponent() {
    ...
    wings = MiddlewareFacade.getDefault();
    wings.addHostDiscoveryPlugin(new BluetoothHDP());
  }
  ...
}
```

2. **Iniciar o *middleware* -** A inicialização do *middleware* é realizada durante a própria inicialização do componente, no método *startImpl*. O *middleware* é inicializado através da invocação do método *startMiddleware*, de acordo com o Código 4.2.

Código 4.2: Inicialização do *middleware Wings*

```
public class DeviceSearcherComponent
  extends FunctionalComponent
  implements HostDiscoveryListener {
  ...
  public void startImpl() {
    wings.startMiddleware();
  }
  ...
}
```

3. **Implementar um *Observer* para a descoberta de nós -** Os clientes do *middleware Wings* são notificados da descoberta de nós através do disparo de eventos. Estes eventos são anunciados seguindo o modelo de notificação do padrão de projeto Observer [GHJV95]. De acordo com este padrão, é necessário implementar uma classe responsável pelo recebimento destes eventos. No caso do *Wings*, a classe de recebimento

dos eventos deve implementar dois métodos definidos na interface *HostDiscoveryListener*: *hostDiscovered* recebe a notificação da descoberta de um dispositivo remoto e *searchingError* recebe a notificação que a busca por dispositivos foi finalizada em virtude da ocorrência de algum erro.

De acordo com a especificação do componente `DeviceSearcher`, o evento `deviceFound` deve ser disparado sempre que um dispositivo for encontrado. O anúncio deste evento é implementado no método *hostDiscovered* da interface *HostDiscoveryListener*; a notificação da descoberta de dispositivos remotos enviada pelo Wings é encapsulada no evento `deviceFound` e anunciada de acordo com o modelo de interação baseado em eventos definido pelo modelo de componentes COMPOR. A notificação de eventos com base no modelo COMPOR é realizada a partir de componentes funcionais. Em virtude disso, o próprio componente `DeviceSearcher` é o ouvinte das notificações de descoberta e, portanto, implementa os métodos da interface *HostDiscoveryListener*.

4. **Iniciar a descoberta de nós -** A inicialização da descoberta de nós no *Wings* é realizada através do método *discoverHosts*. Este método recebe como parâmetro uma instância do tipo *HostDiscoveryListener*, descrito no passo anterior. Uma vez que o próprio componente `DeviceSearcher` é o ouvinte das notificações de descoberta de nós, a palavra-chave *this* é utilizada como parâmetro do método. O método *discoverHosts* é invocado no serviço `searchNearbyDevices`, de acordo com o Código 4.3.

Código 4.3: Inicialização da descoberta de nós

```
public class DeviceSearcherComponent
  extends FunctionalComponent
  implements HostDiscoveryListener {
  ...
  public Long searchNearbyDevices() {
    return new Long(wings.discoverHosts(this));
  }
}
```

Componente `Similarity`

O cálculo de similaridade entre os usuários implementado neste componente leva em consideração o conjunto de interesses dos usuários. Estes interesses são representados por palavras-chave, tais como "futebol", "leitura", "música", etc. Esta implementação em Java utiliza a classe `java.lang.String` para representar cada interesse do usuário e o conjunto de interesses é representado por instâncias do tipo `java.util.Set`. A fórmula utilizada para calcular a similaridade entre os usuários é apresentada na Definição 4.1.

Definição 4.1 *Sejam A e B os conjuntos de interesses de dois usuários distintos, a similaridade S entre os usuários é dada por:*

$$S(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

As seguintes propriedades são aplicáveis ao cálculo da similaridade entre os usuários:

- $S(A, A) = 1$
- $S(A, \emptyset) = 0$

Por exemplo, dados os interesses de dois usuários, representados pelos seguintes conjuntos,

- $A = \{futebol, leitura, samba\}$
- $B = \{viagens, futebol, economia\}$

a similaridade entre eles é dada por:

$$\begin{aligned} S(A, B) &= \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{|\{futebol, leitura, samba\} \cap \{viagens, futebol, economia\}|}{|\{futebol, leitura, samba\} \cup \{viagens, futebol, economia\}|} \\ &= \frac{|\{futebol\}|}{|\{futebol, leitura, samba, viagens, economia\}|} \\ &= \frac{1}{5} \\ &= 0,2 \end{aligned}$$

Neste exemplo a similaridade entre os usuário é pequena, visto que ambos possuem apenas um único interesse comum em um universo de cinco interesses distintos. Caso o número de interesses comuns aumente ou o conjunto de interesses distintos diminua, a similaridade entre os usuários aumentará, uma vez que o cálculo é realizado com base na razão entre a intersecção e a união dos interesses.

Uma vez que o conjunto de interesses do usuário é representado por instâncias do tipo `java.util.Set`, pertencente ao *Java Collections Framework*[3], a implementação do cálculo da similaridade na linguagem Java é realizada através de métodos providos pela própria API deste *framework*. A intersecção entre os conjuntos é obtida através do método *retainAll* e a união entre os conjuntos através do método *addAll*. A cardinalidade dos conjuntos que representam a intersecção e a união é obtida através do método *size*. Esses três métodos são definidos na interface `java.util.Collection`. Por fim, a divisão entre as cardinalidades dos conjunto é calculada e o resultado é retornado através de uma variável do tipo `java.lang.Float`.

Componente `ContentSharing`

Este componente é responsável pela disponibilização de conteúdo para os usuários, possibilitando a disseminação de informação entre os membros das comunidades. O componente `ContentSharing` armazena as informações relacionadas ao conteúdo disponibilizado pelos usuários em um arquivo de propriedades cujo caminho é definido pela propriedade `contentFile`. Os recursos relacionados a um determinado interesse são recuperados através do serviço `getFilesByInterest`. É possível recuperar um arquivo específico através do serviço `getFile`. Novos arquivos são associados a um determinado interesse através do serviço `addFile`.

Implementação

Da mesma forma que o componente `UserManager`, este componente armazena as informações acerca do conteúdo associado a cada interesse do usuário em um arquivo de propriedades com pares chave/valor. A definição e recuperação destas propriedades são realizadas

[3] http://java.sun.com/docs/books/tutorial/collections/index.html

também através da classe `java.util.Properties` da API de Java. A seguir é apresentado um trecho deste arquivo de propriedades.

```
leitura = c:\casmurro.doc, c:\livros.pdf
futebol = c:\fifa.doc
música = c:\historia-da-música.pdf
```

Este arquivo armazena a lista de arquivos referentes ao conteúdo disponibilizado pelo usuário. Para cada interesse do usuário é armazenada a lista de arquivos, que é recuperada e definida através do próprio nome do interesse.

4.5 Sumário do Capítulo

Neste capítulo foi apresentada a infra-estrutura de software para a construção de aplicações para Comunidades Virtuais Móveis. Para a especificação dessa infra-estrutura, foram considerados os requisitos identificados na análise de trabalhos relevantes ao tema de Comunidades Virtuais Móveis. Mediante essa identificação, os requisitos foram agrupados em componentes de software que interagem entre si através dos modelos de interação especificados no modelo de componentes COMPOR. Por fim, com base na especificação dos componentes, foram apresentados detalhes da implementação de cada um dos componentes, tais como os algoritmos e estruturas de dados utilizados.

Capítulo 5

Estudo de Caso

Neste capítulo é apresentado um estudo de caso com o objetivo de validar a utilização da infra-estrutura de software descrita no capítulo anterior. Inicialmente, na Seção 5.1 é apresentada a contextualização e o conjunto de requisitos identificados para a implementação do estudo de caso. Com base nestes requisitos, na Seção 5.2 são apresentadas as modificações realizadas na infra-estrutura e na Seção 5.3 são descritos os passos necessários para a composição da aplicação e a definição das suas propriedades. Por fim, as etapas relacionadas à execução da aplicação assim como as telas de execução do estudo de caso são apresentadas na Seção 5.4.

5.1 Contextualização e Requisitos

A motivação para a implementação deste estudo de caso é baseada no seguinte cenário, situado no Laboratório de Sistemas Embarcados e Computação Pervasiva da Universidade Federal de Campina Grande.

Durante conversas informais ao longo de um dia de trabalho no laboratório surgem discussões sobre temas de interesse comum à maioria dos participantes. As pessoas portam dispositivos móveis que possuem tecnologias de comunicação sem fio e têm a intenção de compartilhar informações sobre o tema em questão que estão disponíveis em seus dispositivos. Além disto, gostariam de relacionar o grupo de pessoas presentes na discussão ao tema em questão, para que possam ser notificadas posteriormente da aproximação de algum dos participantes deste grupo. Desta forma, discussões podem ser estabelecidas mediante novos

encontros, assim como novas informações podem ser disponibilizados. Com isso, o tema discutido é aprofundado com o passar do tempo e o grupo de discussão evolui.

A maioria dos requisitos necessários para a implementação de uma aplicação que dê suporte ao cenário descrito anteriormente é contemplada pela especificação da infra-estrutura de software apresentada no Capítulo 4. O único requisito necessário para a implementação do cenário e que não está presente na especificação da infra-estrutura é o mecanismo utilizado pelo usuário para interagir com os serviços providos pela aplicação. Esta interação com o usuário pode ser realizada através de uma interface gráfica provida pela aplicação. Este requisito pode ser definido então da seguinte forma.

- **Interface gráfica para interação com o usuário** - Os usuários da aplicação devem acessar as informações e serviços disponibilizados pela aplicação através de uma interface gráfica. Esta interface é o meio de interação entre os usuários e a aplicação.

A implementação dos componentes apresentada na Seção 4.4 é suficiente para implementar uma aplicação que dê suporte ao cenário descrito anteriormente. Os serviços oferecidos aos usuários desta aplicação não demandam algoritmos sofisticados e, portanto, podem utilizar a implementação existente dos componentes.

5.2 Instanciação da Infra-estrutura

Com base no requisito adicional de interface gráfica apresentado anteriormente, foi necessário acrescentar um componente à infra-estrutura de software, de acordo com a ilustração da Figura 5.1. O novo componente adicionado — `GUI` — implementa a interface gráfica da aplicação. Ao longo das próximas subseções é apresentada a descrição deste componente e algumas características inerentes à sua implementação. A especificação completa é apresentada no Apêndice B.

Descrição do Componente de Interface Gráfica

Este componente é responsável pela camada de apresentação da aplicação do estudo de caso, possibilitando que o usuário utilize os serviços disponibilizados pela infra-estrutura através de uma interface gráfica. A interação com a infra-estrutura é realizada através da

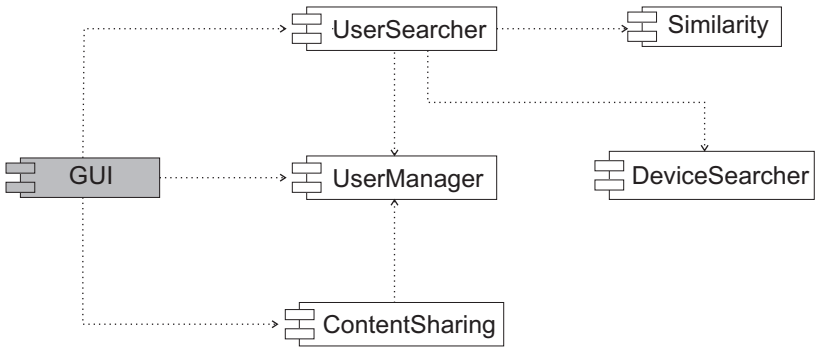

Figura 5.1: Componente adicionado para o estudo de caso

utilização dos serviços e eventos dos componentes `UserSearcher`, `UserManager` e `ContentSharing`, de acordo com as dependências ilustradas na Figura 5.1. Na Figura 5.2 é apresentada a especificação deste componente, com a definição dos serviços requeridos e dos eventos de interesse.

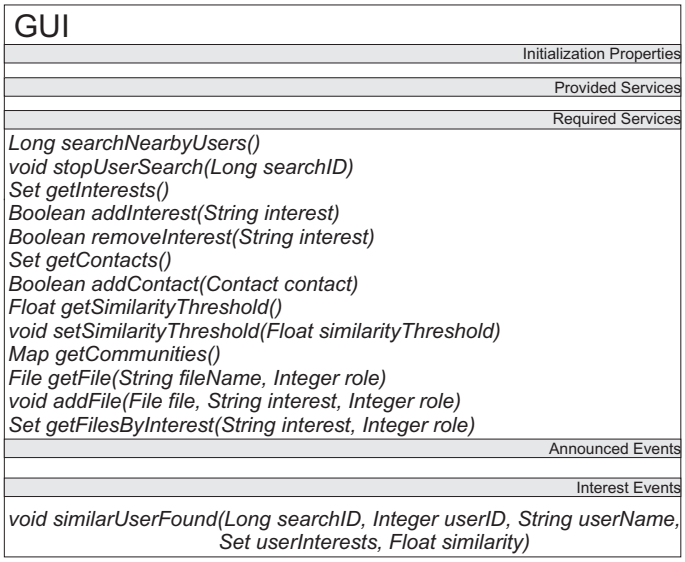

Figura 5.2: Especificação do componente de interface gráfica

O componente `GUI` utiliza o componente `UserSearcher` para a inicialização e interrupção da busca de usuários (serviços `searchNearbyUsers` e `stopUserSearch`) e para receber a notificação da proximidade de usuários similares (evento `similaruserFound`). O componente `UserManager` é utilizado para recuperar as informações que são apresentadas para o usuário, assim como alterá-las de acordo com as requisições do usuário. São recuperados/alterados os interesses (serviços `getInterests`,

`addInterest` e `removeInterest`), lista de contatos (serviços `getContacts` e `addContact`), comunidades (serviço `getCommunities`) e limiar de similaridade do usuário (serviços `getSimilarityThreshold` e `setSimilarityThreshold`). Por fim, o componente `ContentSharing` é utilizado para compartilhar os arquivos do usuário em uma determinada comunidade (serviço `addFile`) e para recuperar arquivos compartilhados por outros usuários (serviços `getFile` e `getFilesByInterest`).

Implementação do Componente de Interface Gráfica

A implementação deste componente utiliza o *Thinlet toolkit*[1] para a construção da interface gráfica. Este *toolkit* disponibiliza uma classe Java que interpreta a definição dos componentes gráficos e suas propriedades a partir de um arquivo XML. Desta forma, existe uma separação entre a apresentação gráfica descrita em um arquivo XML e os métodos da aplicação definidos através de código Java. A utilização deste *toolkit* facilita a construção de protótipos, uma vez que a quantidade de código necessária para definir a interface gráfica é bem menor que a de arcabouços para a construção de interface gráfica em Java, tais como Swing[2] e AWT[3].

Os arquivos XML com a definição dos componentes gráficos da aplicação são interpretados durante a inicialização do componente, no método *startImpl*, cuja implementação é apresentada no Código 5.1. Especificamente durante a instanciação da classe *MVCThinlet* (linha 4), os arquivos XML são interpretados e os componentes gráficos são criados. Após a interpretação e criação dos componentes gráficos, a interface é exibida através da instanciação da classe *MVCFrameLauncher* (linha 5). Em seguida, o componente `GUI` inicia a busca por usuários através da requisição do serviço `searchNearbyUsers` (linha 7).

Código 5.1: Inicialização do componente de interface gráfica

```
1 public class GUIComponent extends FunctionalComponent {
2   ...
3   public void startImpl() {
4     thinlet = new MVCThinlet(this);
```

[1] `http://www.thinlet.com/`
[2] `http://www.java.sun.com/docs/books/tutorial/uiswing/`
[3] `http://www.oreilly.com/catalog/javawt/book/`

```
5       new MVCFrameLauncher(thinlet, this);
6       ServiceResponse response =
7         super.doIt(new ServiceRequest("searchNearbyUsers"));
8       ...
9     }
10    ...
11  }
```

5.3 Composição da Aplicação e Definição das Propriedades de Inicialização

Uma vez que os componentes específicos para o estudo de caso foram definidos e implementados, a composição da aplicação pode ser realizada. São utilizados os cinco componentes que fazem parte da infra-estrutura e o novo componente específico do estudo de caso.

Como apresentado no Capítulo 3, para realizar a composição de uma aplicação utilizando o arcabouço de componentes COMPOR, é necessário inicialmente especificar a estrutura hierárquica da aplicação, com a definição dos contêineres nos quais os componentes funcionais são agrupados. A estrutura hierárquica da aplicação do estudo de caso é ilustrada na Figura 5.3. Os componentes que fazem parte da infra-estrutura são agrupados no contêiner `Mobile Virtual Communities` e os componentes específicos do estudo de caso são agrupados no contêiner `Application`. Por fim, estes dois contêineres são agrupados no contêiner `RootContainer`.

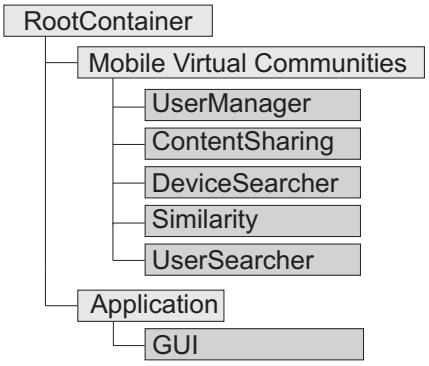

Figura 5.3: Hierarquia da aplicação com base no modelo de componentes COMPOR

Com base na definição da estrutura hierárquica é possível codificar a composição da aplicação utilizando o arcabouço de componentes COMPOR. No Código 5.2 é apresentada a implementação em Java que compõe a aplicação de acordo com a estrutura hierárquica definida anteriormente.

Código 5.2: Composição da aplicação

```
1  public class RootContainer extends ScriptContainer {
2    public RootContainer() {
3      super("RootContainer");
4      Container mvcContainer =
5        new Container("Mobile Virtual Communities");
6
7      UserManagerComponent userManagerComponent =
8        new UserManagerComponent();
9      userManagerComponent.setInitializationPropertyValue(
10       UserManagerComponent.PROPERTY_DATA_FILE,
11       new File("/userData.txt"));
12
13     ContentSharingComponent contentSharingComponent =
14       new ContentSharingComponent();
15     contentSharingComponent.setInitializationPropertyValue(
16       ContentSharingComponent.PROPERTY_CONTENT_FILE,
17       new File("/contentFile.txt"));
18
19     mvcContainer.addComponent(userManagerComponent);
20     mvcContainer.addComponent(contentSharingComponent);
21     mvcContainer.addComponent(new DeviceSearcherComponent());
22     mvcContainer.addComponent(new SimilarityComponent());
23     mvcContainer.addComponent(new UserSearcherComponent());
24
25     Container appContainer = new Container("Application");
26     appContainer.addComponent(new GUIComponent());
27
```

```
28          super.addComponent(mvcContainer);
29          super.addComponent(appContainer);
30      }
31  }
```

A composição da aplicação é realizada na classe que implementa o contêiner que representa a raiz da hierarquia (*RootContainer*). Esta classe estende a classe *ScriptContainer* (linha 1), que representa um contêiner especial utilizado para a raiz da hierarquia de aplicações baseadas no arcabouço de componentes COMPOR. Na linha 3 é definido o nome do contêiner raiz. Os contêineres `Mobile Virtual Communities` e `Application` são criados nas linhas 5 e 25, respectivamente. Após a criação dos contêineres, os componentes funcionais são instanciados e adicionados aos respectivos contêineres através do método *addComponent* (linhas 8, 14, 19-23 e 26). Antes da adição do componente `UserManager` ao contêiner, a propriedade de inicialização `dataFile` é definida (linhas 9-11); o arquivo que contém os dados do usuário é armazenado na raiz do sistema de arquivos, com o nome *userData.txt*. Antes da adição do componente `ContentSharing` ao contêiner, a propriedade de inicialização `contentFile` é definida (linhas 15-17); o arquivo que contém as informações dos recursos é armazenado na raiz do sistema de arquivos, com o nome *contentFile.txt*.

Visto que os contêineres `Mobile Virtual Communities` e `Application` estão criados e seus componentes estão adicionados, os contêineres são então adicionados ao contêiner `RootContainer` (linhas 28-29), que representa a raiz da hierarquia de componentes. A adição de contêineres é também realizada através do método *addComponent*, visto que os contêineres e os componentes funcionais implementam uma interface comum (*AbstractComponent*), de acordo com o padrão de projeto *Composite* apresentado na Seção 3.3.

Com base no código apresentado anteriormente, a aplicação está preparada para a execução, visto que no contêiner que representa a raiz da hierarquia estão adicionados os seus *contêineres-filhos*, que por sua vez também estão com os seus *componentes-filhos* adicionados. Além disto, as propriedades de inicialização dos componentes também estão definidas.

5.4 Execução da Aplicação

Três passos são necessários para a execução da aplicação: instanciação do contêiner que representa a raiz da hierarquia, instanciação do *script* de execução da aplicação e a inicialização deste script. Estes passos estão implementados na classe *MainClass*, cujo método *main* é apresentado no Código 5.3.

Código 5.3: Inicialização da aplicação

```
1 public class MainClass {
2   public static void main(final String[] _args) {
3     RootContainer rootContainer = new RootContainer();
4     ExecutionScript executionScript =
5       new ExecutionScript(rootContainer);
6     executionScript.init();
7   }
8 }
```

Na linha 3 é instanciada a classe *RootContainer*, cujo código foi apresentado na Seção 5.3 e na qual é implementada a representação da hierarquia da aplicação. Na linha 5 é instanciado o *script* de execução da aplicação, implementado pela classe *ExecutionScript*. Este *script* possui uma referência para o *rootContainer* para que possa ter acesso aos contêineres e componentes funcionais que fazem parte da aplicação. A execução da aplicação é iniciada após a invocação do método *init* (linha 6) do *script* de execução. Este método inicia os componentes e contêineres seguindo a ordem na qual foram adicionados ao *rootContainer*.

Primeiramente são iniciados os componentes do contêiner `Mobile Virtual Communities` e em seguida os componentes do contêiner `Application`. A ordem na qual os contêineres foram adicionados é intencional; o contêiner `Mobile Virtual Communities` é adicionado antes do contêiner `Application`, pois os componentes da infra-estrutura devem estar disponíveis (inicializados) para receber invocações dos componentes da aplicação.

Os componentes são também inicializados de acordo com a ordem em que foram adicionados aos contêineres. O contêiner `Mobile Virtual Communities` inicia o componente `UserManager`, `ContentSharing`, `DeviceSearcher`, `Similarity` e por

fim `UserSearcher`; a partir deste momento os serviços e eventos da infra-estrutura podem ser utilizados pelos componentes da aplicação. O contêiner `Application` inicia o componente `GUI`; após a inicialização deste último componente, a interface gráfica da aplicação é exibida.

Ao longo do restante desta seção são apresentadas as telas da execução da aplicação do estudo de caso. Para cada uma das telas são descritas as funcionalidades disponibilizadas para o usuário.

Edição dos Interesses do Usuário

O usuário pode editar seus interesses de acordo com a tela ilustrada na Figura 5.4. Novos interesses podem ser definidos na caixa de texto *Interesse* e adicionados através do botão de adição. Após a adição de um novo interesse, a lista de interesses é atualizada. Interesses também podem ser excluídos através da seleção na lista e utilização do botão de exclusão. Após a exclusão de um interesse, a lista de interesses também é atualizada.

Figura 5.4: Tela para a edição dos interesses do usuário

Visualização da Lista de Contatos

Na Figura 5.5 são ilustradas as telas utilizadas para a visualização da lista de contatos do usuário. O usuário pode visualizar a sua lista de contatos de acordo com a tela apresentada na Figura 5.5(a). O nome de cada usuário da sua lista de contatos é exibido na lista de contatos. Através do botão *Visualizar detalhes do contato* o usuário pode visualizar informações mais

detalhadas do contato. Estas informações são apresentadas de acordo com a tela ilustrada na Figura 5.5(b). São exibidos o nome, lista de interesses e a similaridade existente entre o usuário e o contato.

(a) Lista de contatos (b) Detalhes de um contato

Figura 5.5: Telas para a visualização da lista de contatos do usuário

Notificação da Proximidade de Usuários Similares

A tela ilustrada na Figura 5.6 é exibida sempre que um novo usuário que possui interesses similares ao usuário que utiliza o dispositivo é encontrado pelo componente que notifica a proximidade de indivíduos. Com base nesta notificação, o usuário pode adicionar este novo usuário à sua lista de contatos, através do botão *Sim*. O botão *Não* é utilizado caso o usuário não deseje adicionar o novo usuário à lista de contatos.

Figura 5.6: Tela de notificação da proximidade de usuários similares

Visualização das Comunidades

O usuário também pode visualizar a lista de comunidades das quais é membro (Figura 5.7). O nome de cada comunidade é apresentado na lista *Comunidades*, assim como o número de contatos que participam de cada comunidade. Os membros de cada comunidade são exibidos na lista de *Usuários*.

Figura 5.7: Tela para a visualização da lista de comunidades

Acesso ao Conteúdo Disponibilizado nas Comunidades

O usuário pode visualizar a lista de arquivos disponibilizados em cada uma de suas comunidades (Figura 5.8). O nome de cada comunidade é apresentado na lista *Comunidades*. Os arquivos associados a cada uma das comunidades são exibidos na lista de *Arquivos*. A visualização do arquivo é realizada através de um clique duplo no nome do arquivo.

Ajuste do Limiar de Similaridade

O usuário pode ajustar o limiar de similaridade que deseja utilizar para receber a notificação de usuário similares, evitando desta forma a notificação de usuários indesejados. Este ajuste é realizado através do *slider Limiar de similaridade dos usuários*, ilustrado na Figura 5.9. O botão *Salvar* é utilizado para persistir o valor do limiar de similaridade.

Figura 5.8: Tela para a visualização do conteúdo disponibilizado nas comunidades

Figura 5.9: Tela para o ajuste do limiar de similaridade

Capítulo 6

Conclusões e Trabalhos Futuros

Três características estão presentes nos dispositivos móveis atuais: a *conectividade* disponibilizada pelas tecnologias de rede sem fio, a *mobilidade* obtida com a redução do tamanho dos dispositivos e a *capacidade computacional* obtida com os avanços no desenvolvimento de componentes eletrônicos, tais como processadores e memória. A união destas três características vem permitindo o desenvolvimento de aplicações mais complexas, dentre elas aplicações que promovem a formação de Comunidades Virtuais Móveis entre os usuários que utilizam tais dispositivos. Estas comunidades são constituídas por indivíduos que possuem interesses comuns e que utilizam os mecanismos de comunicação disponibilizados pelas aplicações para interagir com outros membros da comunidade.

Estas aplicações para Comunidades Virtuais Móveis possuem características comuns que são compartilhadas pela maioria das aplicações para este domínio. Entretanto, não existem arcabouços de software que contemplem estas características de forma integrada e conseqüentemente facilitem o desenvolvimento de tais aplicações. Desta forma, é necessário implementar estas funcionalidades para cada nova aplicação desenvolvida.

Mediante esta necessidade, este trabalho apresentou uma infra-estrutura de software baseada em componentes com o intuito de facilitar o desenvolvimento de aplicações para Comunidades Virtuais Móveis. Esta infra-estrutura disponibiliza um conjunto de componentes, baseados no arcabouço COMPOR, que permitem a reutilização de funcionalidades de Comunidades Virtuais Móveis em diversas aplicações. Os componentes da infra-estrutura disponibilizam serviços e eventos para a notificação da aproximação de dispositivos e indivíduos, o cálculo da similaridade entre usuários, representação dos interesses do usuário, ge-

rência do acesso às informações do usuário e disponibilização de conteúdo nas comunidades.

A infra-estrutura apresentada neste trabalho é composta por uma especificação e uma implementação. Na especificação são descritos os componentes, através dos seus serviços e eventos; a dependência entre os componentes; e a interação existente entre eles. Neste trabalho é apresentada também uma implementação para cada um dos componentes da infra-estrutura, utilizando como base a plataforma Java ME CDC.

Além disto, a infra-estrutura desenvolvida oferece suporte para a composição dinâmica de aplicações, possibilitando a adição, remoção e troca de componentes em tempo de execução. Por exemplo, um novo mecanismo para o cálculo da similaridade entre os usuários pode ser acrescentado à aplicação através da adição de um novo componente de similaridade e remoção do anterior. Requisitos não previstos durante a concepção da aplicação também podem ser acrescentados com facilidade, visto que a infra-estrutura também possui suporte para evolução de software não antecipada.

Como forma de validar a utilização da infra-estrutura, apresentou-se neste trabalho um estudo de caso. Para este estudo de caso foi implementado um componente específico para permitir a interação do usuário com os serviços disponibilizados pela aplicação, através de uma interface gráfica.

Algumas discussões sobre trabalhos futuros são apresentadas a seguir.

Integração com *Personal Information Manager*

A maioria dos dispositivos móveis armazena informações da lista de contados do usuário, tais como nomes, endereços e números de telefone, além de informações da agenda do usuário, tais como compromissos e lembretes. Todas estas informações podem ser acessadas através de softwares conhecidos como *Personal Information Managers* (PIMs). Estas informações também podem ser sincronizadas com o computador pessoal do usuário e acessadas através de aplicativos de softwares PIM, tais como Outlook[1] e Evolution[2].

A plataforma Java ME possui uma especificação denominada *Personal Information Management Optional Package*[3], que disponibiliza um conjunto de APIs para facilitar o acesso e

[1] `http://www.microsoft.com/outlook/`
[2] `http://www.gnome.org/projects/evolution/`
[3] `http://jcp.org/en/jsr/detail?id=75`

edição deste tipo de informação. Com base nestas APIs é possível extender a infra-estrutura de componentes para oferecer novos serviços para o usuário, tais como calendários compartilhados nas comunidades e sincronização da lista de contatos do usuário com a agenda telefônica.

Interface Adaptável às Características do Dispositivo

Vários são os modelos de dispositivos móveis existentes, com configurações de memória, processamento e *display* diferentes. Em virtude desta diversidade, a interface gráfica das aplicações poderia ser projetada para que seu conteúdo se ajustasse às características de cada dispositivo, evitando desta forma a redefinição da interface para cada dispositivo. Para que isso seja possível é necessária a utilização de linguagens que descrevam as características do dispositivo, tais como tamanho da tela, *content types* suportados, entre outras.

O *World Wide Web Consortium*[4] (W3C) e a *Open Mobile Alliance*[5] (OMA) possuem projetos que utilizam linguagens baseadas em XML para descrever dispositivos. São eles: *Composite Capabilities/Preference Profiles*[6] (CC/PP), *Device Independence*[7] e *User Agent Profile*[8] (UAProf). Estas linguagens baseadas em XML destinadas à descrição do perfil de dispositivos móveis podem ser utilizadas no componente de interface gráfica da infra-estrutura apresentada neste trabalho, com o objetivo de permitir a adaptação dos componentes gráficos de acordo com as características de cada dispositivo.

Definição de Ontologias para a Representação dos Interesses do Usuário

A implementação do componente `UserManager` apresentada neste trabalho considera apenas palavras-chave isoladas para a representação dos interesses do usuário. Um mecanismo mais robusto para a representação destes interesses pode ser a utilização de ontologias [GPFLC04].

Com base na definição de ontologias, é possível representar as relações existentes os usuários, assim como inferir e obter fatos acerca do usuário e com isso adicionar novas

[4] `http://www.w3.org/`
[5] `http://www.openmobilealliance.org/`
[6] `http://www.w3.org/Mobile/CCPP/`
[7] `http://www.w3.org/2001/di/`
[8] `http://www.openmobilealliance.org/release_program/uap_v2_0.html`

informações ao seu perfil [PCP+04]. Atualmente está sendo desenvolvido um trabalho no Laboratório de Sistemas Embarcados e Computação Pervasiva cujo objetivo é investigar mecanismos para a aquisição, representação e disponibilização de informações contextuais, através de um arcabouço baseado em ontologias.

Caso sejam utilizadas ontologias na infra-estrutura apresentada neste trabalho, é necessário modificar a representação dos interesses e o cálculo da similaridade entre os usuários. Mais especificamente, os componentes `UserManager` e `Similarity`, detalhados no Capítulo 4, devem ser alterados caso estas modificações sejam realizadas.

Implementação da Infra-estrutura para a Plataforma Java ME CLDC

Discutir a implementação dos componentes da infra-estrutura para a configuração Java ME CLDC com o intuito de possibilitar a implantação das aplicações para Comunidades Virtuais Móveis em um conjunto maior de dispositivos portáteis. É importante ressaltar que a implementação dos componentes nesta plataforma deve considerar as restrições de memória e processamento dos dispositivos, assim como as limitações impostas pelas próprias APIs. Além disto, a característica de composição de dinâmica de software não seria contemplada pela versão CLDC.

Adição de Novos Componentes à Infra-estrutura

Analisar outras aplicações e arcabouços relacionados ao domínio de Comunidades Virtuais Móveis com o objetivo de identificar novos requisitos que possam ser adicionados à infra-estrutura de software para este domínio de aplicação. O acréscimo de novas funcionalidades à infra-estrutura é facilitado, uma vez que o modelo de componentes no qual a infra-estrutura é baseada possui suporte para a evolução de software não antecipada.

Infra-estrutura Baseada em Sistemas Multi-Agentes

As aplicações para Comunidades Virtuais Móveis podem ser projetadas utilizando os conceitos da concepção de Sistemas Multi-Agentes [Wei00]. Para isso, a análise de tais aplicações deve ter foco na *interação* entre os indivíduos (*agentes*) que fazem parte das comunidades (*organizações*). Além destes conceitos de interação, organização e agente, outros conceitos,

tais como protocolos que regem as interações entre os agentes [FAPC04], papéis de interação, mecanismos de governança [CAG+06], ambiente, entre outros, podem ser utilizados para a concepção destas aplicações.

Alguns trabalhos têm sido propostos com foco no desenvolvimento de plataformas multi-agentes para a construção de aplicações de comunidades [CMA04; MPS99]. Um outro tópico interessante a ser investigado é a definição de uma metodologia baseada em agentes para o desenvolvimento de aplicações para Comunidades Virtuais Móveis, definindo um mapeamento entre os conceitos utilizados na área de Comunidade Virtuais Móveis e aqueles empregados na área de Sistemas Multi-Agentes. A infra-estrutura de componentes apresentada neste trabalho poderia ser utilizada no desenvolvimento das aplicações projetadas com base na metodologia definida.

Bibliografia

[Ale04] B. Alexander. Going Nomadic: Mobile Learning in Higher Education. *EDUCAUSE Review*, 39(5):28–35, Setembro/Outubro 2004.

[Alm04] H. O. Almeida. COMPOR - Desenvolvimento de Software para Sistemas Multiagentes. Dissertação de Mestrado, Mestrado em Informática da Universidade Federal de Campina Grande, Campina Grande, PB, Março 2004.

[ANG02] R. Aldunate, M. Nussbaum, and R. Gonzales. An Agent-Based Middleware for Supporting Spontaneous Collaboration among Co-located, Mobile, and not Necessarily Known People. In *Workshop on AD HOC Communications and Collaboration in Ubiquitous Computing Environments*, Nova Orleans, EUA, Novembro 2002.

[APF+06] H. O. Almeida, A. Perkusich, G. V. Ferreira, E. C. Loureiro Filho, and E. B. Costa. A Component Model to Support Dynamic Unanticipated Software Evolution. In *Proceedings of International Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pages 262–267, San Francisco, EUA, 2006.

[APPC04] H. O. Almeida, A. Perkusich, R. B. Paes, and E. B. Costa. Composição Dinâmica de Componentes para Aplicações com Mudanças Freqüentes de Requisitos. In *Anais do IV Workshop de Desenvolvimento Baseado em Componentes*, volume 4, pages 9–14, João Pessoa, PB, 2004.

[BBB+00] F. Bachmann, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, and K. Wallnau. Technical Concepts of Component-based Soft-

ware Engineering. Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon - Software Engeneering Institute, Maio 2000.

[BCGS04] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovi, editors. *Mobile Ad Hoc Networking*. Wiley-IEEE Press, Hoboken, EUA, Agosto 2004.

[BGF05] V. Bryl, P. Giorgini, and S. Fante. ToothAgent: A Multi-Agent System for Virtual Communities Support. Technical Report DIT-05-064, University of Trento, Department of Information and Communication Technology, Trento, Itália, 2005.

[BRJ00] G. Booch, J. Rumbaugh, and I. Jacobson. *UML - Guia do Usuário*. Campus, Rio de Janeiro, RJ, 2000.

[CAG+06] G. Carvalho, H. Almeida, M. Gatti, G. Ferreira, R. Paes, A. Perkusich, and C. Lucena. Dynamic Law Evolution in Governance Mechanisms for Open Multi-Agent Systems. In *Second Workshop on Software Engineering for Agent-oriented Systems - SEAS*, 2006. Aceito para publicação.

[CMA04] L. M. Camarinha-Matos and H. Afsarmanesh. A Multi-agent Based Infrastructure to Support Virtual Communities in Eelderly Care. *International Journal of Networking and Virtual Organisations*, 2(3):246–266, 2004.

[CRI+01] J. M. Carroll, M. B. Rosson, P. L. Isenhour, C. H. Ganoe, D. R. Dunlap, J. Fogarty, W. A. Schafer, and C. A. Van Metre. Designing our Town: MOOsburg. *International Journal of Human-Computer Studies*, 54(5):725–751, 2001.

[DCC03] F. M. O. Dias, M. A. Casanova, and M. T. M. Carvalho. Workflow Execution in Disconnected Environments. In *XVIII Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 229–239, Manaus, AM, Outubro 2003.

[FAPC04] G. V. Ferreira, H. O. Almeida, A. Perkusich, and E. B. Costa. Especificação e Implementação de Protocolos de Interação entre Agentes para a Plataforma COMPOR. *Infocomp Revista de Ciência da Computação*, 3(2):1–7, Novembro 2004.

[FICR02] U. Farooq, P. L. Isenhour, J. M. Carroll, and M. B. Rosson. MOOsburg++: Moving Towards a Wireless Virtual Community. In *Proceedings of the 2002 International Conference on Wireless Networks*. CSREA Press, 2002.

[Fla05] D. Flanagan. *Java In A Nutshell*. O'Reilly, Sebastopol, EUA, 5 edition, 2005.

[FLN+04] G. V. Ferreira, E. C. Loureiro Filho, W. F. A. Nogueira, A. F. A. Gomes, H. O. Almeida, and A. Frery. Uma Abordagem Baseada em Componentes para a Construção de Edifícios Virtuais . In *Proceedings of VII Symposium on Virtual Reality - SVR 2004*, volume 7, pages 279–290, São Paulo, 2004. Vida & Consciência.

[Fre00] D. Fremaux. The Next VAS Generation. `http://horizontest.bvdep.com/telecom/default.asp?journalid=2&func=articles&page=0009i31&year=2000&month=9`, Setembro 2000. Acessado em fevereiro de 2006.

[FTF03] N. Fremuth, A. Tasch, and M. Fränkle. Mobile Communities — New Business Opportunities for Mobile Network Operators? In *Proceedings of the 2nd Interdisciplinary World Congress on Mass Customization and Personalization (MCPC)*, 2003.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, Boston, EUA, 1995.

[GK04] T. Goh and D. Kinshuk. Getting Ready For Mobile Learning. In L. Cantoni and C. McLoughlin, editors, *Proceedings of ED-MEDIA 2004 - World Conference on Educational Multimedia, Hypermedia & Telecommunications*, pages 56–63, Lugano, Suíça, Junho 2004. AACE.

[GPFLC04] A. Gomez-Perez, M. Fernandez-Lopez, and O. Corcho. *Ontological Engineering*. Advanced Information and Knowledge Processing. Springer-Verlag, Berlim, Alemanha, 2004.

[GSB+04] W. G. Griswold, P. Shanahan, S. W. Brown, R. Boyer, M. Ratto, R. B. Shapiro, and T. M. Truong. ActiveCampus: Experiments in Community-Oriented Ubiquitous Computing. *Computer*, 37(10):73–81, Outubro 2004.

[HC01] G. T. Heineman and W. T. Councill. *Component Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, Boston, EUA, 2001.

[Hou01] A. Houaiss. *Dicionário Houaiss da Língua Portuguesa*. Objetiva, Rio de Janeiro, RJ, Janeiro 2001.

[HOY+99] F. Hattori, T. Ohguro, M. Yokoo, S. Matsubara, and S. Yoshida. Socialware: Multiagent Systems for Supporting Network Communities. *Communications of the ACM*, 42(3):55–61, Março 1999.

[HSK04] M. Hazas, J. Scott, and J. Krumm. Location-Aware Computing Comes of Age. *Computer*, 37(2):95–97, Fevereiro 2004.

[Int02] International Organization for Standardization. Information Technology - Open Distributed Processing - Reference Model - Enterprise Language. Technical Report ISO/IEC 15414:2002, International Organization for Standardization, Genebra, Suíca, 2002.

[IRC01] P. L. Isenhour, M. B. Rosson, and J. M. Carroll. Supporting Interactive Collaboration on the Web with CORK. *Interacting with Computers*, 13(6):655–676, 2001.

[KADL02] H. Krcmar, Y. Arnold, M. Daum, and J. M. Leimeister. Virtual Communities in Health Care: The Case of "krebsgemeinschaft.de". *SIGGROUP Bulletin*, 23(3):18–23, Dezembro 2002.

[KSP+01] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall. When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks. In *1st International Conference on Peer-to-Peer Computing (P2P 2001)*, pages 75–91, Linköping, Suécia, Agosto 2001. IEEE Computer Society.

[Lau04] K-K. Lau, editor. *Component-Based Software Development: Case Studies*, volume 1 of *Series on Component-Based Software Development*. World Scientific Publishing Company, Cingapura, 2004.

[LDK02] J.M. Leimeister, M. Daum, and H. Krcmar. Mobile Virtual Communities: An Approach to Community Engineering for Cancer Patients. In *Proceedings of the Xth European Conference on Information Systems (ECIS)*, pages 1626–1637, 2002.

[Lex06] Lexico Publishing Group. Dictionary.com. http://dictionary.reference.com/, 2006. Acessado em setembro de 2006.

[Lou06] E. C. Loureiro Filho. Um Middleware Extensível para Disponibilização de Serviços em Ambientes Pervasivos. Dissertação de Mestrado, Mestrado em Informática da Universidade Federal de Campina Grande, Campina Grande, PB, Agosto 2006.

[MPS99] A. Mamdani, J. Pitt, and K. Stathis. Connected Communities from the Standpoint of Multi-agent Systems. *New Generation Computing*, 17(4):381–393, 1999.

[PCP+04] R. Prestes, G. Carvalho, R. Paes, C. Lucena, and M. Endler. Applying Ontologies in Open Mobile Systems. In *OOPSLA'04 Workshop on Building Software for Pervasive Computing*, Vancouver, Canadá, Outubro 2004.

[Rhe93] H. Rheingold. *The Virtual Community: Homesteading on the Electronic Frontier*. Addison-Wesley, Boston, EUA, 1993.

[Rhe03] H. Rheingold. Mobile Virtual Communities. `http://www.thefeaturearchives.com/topic/Culture/Mobile_Virtual_Communities.html`, Julho 2003. Acessado em setembro de 2006.

[RLZ00] A. Rakotonirainy, S. W. Loke, and A. Zaslavsky. Towards Multi-Agent Support for Open Mobile Virtual Communities. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI 2000)*, pages 127–133, Las Vegas, EUA, 2000. CSREA Press.

[SCG99] S. Sheppherd, D. Charmock, and B. Gann. Helping Patients Access High Quality Health Information. *British Medical Journal*, 319:764–766, Setembro 1999.

[SM02] Y. Sumi and K. Mase. Conference Assistant System for Supporting Knowledge Sharing in Academic Communities. *Interacting with Computers*, 14(6):713–737, Dezembro 2002.

[Sun95] Sun Microsystems. The Java Tutorial. `http://java.sun.com/docs/books/tutorial/`, 1995. Acessado em setembro de 2006.

[Sun05] Sun Microsystems. CDC: Java™Platform Technology for Connected Devices. `http://java.sun.com/products/cdc/wp/cdc-whitepaper.pdf`, Junho 2005. Acessado em setembro de 2006.

[Wei00] G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1 edition, Julho 2000.

[WUK99] G. Wang, L. Ungar, and D. Klawitter. Component Assembly for OO Distributed Systems. *Computer*, 32(7):71–78, Julho 1999.

[YKOK03] S. Yoshida, K. Kamei, T. Ohguro, and K. Kuwabara. Shine: a Peer-to-peer Based Framework of Network Community Support Systems. *Computer Communications*, 26(11):1199–1209, Julho 2003.

[Yua03] M. J. Yuan. *Enterprise J2ME: Developing Mobile Java Applications*. Prentice Hall PTR, Indianápolis, EUA, 2003.

Apêndice A

Especificação dos Componentes da Infra-estrutura

Neste apêndice são apresentadas as especificações dos componentes que constituem a infra-estrutura para a construção de aplicações para Comunidades Virtuais Móveis apresentada neste trabalho. Os seguintes tópicos fazem parte da especificação de cada componente: descrição, propriedades de inicialização, serviços providos, serviços requeridos, eventos anunciados e eventos de interesse.

A.1 Componente `UserManager`

- **Descrição -** Gerencia o acesso às informações do usuário: nome, lista de interesses, lista de contatos, limiar de similaridade e comunidades.
- **Propriedades de Inicialização**
  - `File dataFile`

    Esta propriedade de inicialização mantém referência para o arquivo-texto que armazena as informações do usuário, tais como nome, limiar de similaridade, lista de contatos, interesses e comunidades.
- **Serviços Providos**
  - `Boolean addInterest(String interest)`

Adiciona o interesse `interest` à lista de interesses do usuário. Este serviço retorna `true` caso a lista de interesses do usuário não possua o interesse `interest`; caso contrário, retorna `false`.

- `Boolean removeInterest(String interest)`

  Remove o interesse `interest` da lista de interesses do usuário. Este serviço retorna `true` caso a lista de interesses do usuário já possua o interesse `interest`; caso contrário, retorna `false`.

- `Set getInterests()`

  Recupera a lista de interesses do usuário. Este serviço retorna um `Set` de `String`s com os interesses do usuário.

- `Boolean addContact(Contact contact)`

  Adiciona o contato `contact` à lista de contatos do usuário. Este serviço retorna `true` caso a lista de contatos do usuário não possua o contato `contact`; caso contrário, retorna `false`.

- `Set getContacts()`

  Recupera a lista de contatos do usuário. Este serviço retorna um `Set` de `Contact`s com os contatos do usuário.

- `Boolean isInContactList(Integer userID)`

  Verifica se o usuário identificado por `userID` está na lista de contatos do usuário em questão. Este serviço retorna `true` caso a lista de contatos do usuário possua o contato identificado por `userID`; caso contrário, retorna `false`.

- `void setSimilarityThreshold(Float similarityThreshold)`

  Define o limiar de similaridade do usuário de acordo com o valor do parâmetro `similarityThreshold`.

- `Float getSimilarityThreshold()`

  Recupera o limiar de similaridade definido pelo usuário. Este serviço retorna um `Float` que indica o limiar de similaridade definido pelo usuário.

- `Map getCommunities()`

Recupera a lista de comunidades do usuário. Este serviço retorna um `Map` cuja chave é uma `String` que representa um interesse e cujo valor é um `Set` de `Contacts` com os contatos associados ao interesse em questão.

- **Serviços Requeridos -** Não possui serviços requeridos.
- **Eventos Anunciados -** Não possui eventos anunciados.
- **Eventos de Interesse -** Não possui eventos de interesse.

A.2 Componente `UserSearcher`

- **Descrição -** Busca usuários próximos a um determinado usuário de acordo com a similaridade existente entre eles. Requer serviços de busca de dispositivos próximos e cálculo de similaridade entre os interesses de usuários.
- **Propriedades de Inicialização -** Não possui propriedades de inicialização.
- **Serviços Providos**
  - `Long searchNearbyUsers()`

    Busca usuários próximos a um determinado usuário. Retorna um `Long` indicando o identificador da busca. Este identificador pode ser utilizado para interromper a busca, através do serviço `stopUserSearch`. Este serviço utiliza o serviço requerido `searchNearbyDevices` para realizar a busca por dispositivos próximos ao dispositivo do usuário em questão.
  - `void stopUserSearch(Long searchID)`

    Interrompe a busca por usuários identificada por `searchID`.
- **Serviços Requeridos**
  - `Float evaluateSimilarity(Set userInterests, Set otherUserInterests)`

    Este componente requer um serviço que calcule a similaridade entre os interesses de dois usuários: `userInterests` e `otherUserInterests`. Ambos os

interesses são definidos através de um `Set` de palavras-chave, representadas por `String`. Retorna um `Float` indicando a similaridade entre os interesses.

- `Long searchNearbyDevices()`

  Este componente requer um serviço que busque dispositivos próximos ao dispositivo do usuário. Retorna um `Long` indicando o identificador da busca. Este identificador pode ser utilizado para interromper a busca, através do serviço requerido `stopDeviceSearch`. Sempre que um dispositivo é encontrado através deste serviço, o evento de interesse `deviceFound` é anunciado.

- `void stopDeviceSearch(Long searchID)`

  Este componente requer um serviço que interrompe a busca por dispositivos identificada por `searchID`.

- `Set getInterests()`

  Este componente requer um serviço que recupere a lista de interesses do usuário. Retorna um `Set` de `String`s com os interesses do usuário.

- `Set getContacts()`

  Este componente requer um serviço que recupere a lista de contatos do usuário. Retorna um `Set` de `Contact`s com os contatos do usuário.

- `Boolean isInContactList(Integer userID)`

  Este componente requer um serviço que verifique se o usuário identificado por `userID` está na lista de contatos do usuário em questão. Retorna `true` caso a lista de contatos do usuário possua o contato identificado por `userID`; caso contrário, retorna `false`.

- `Float getSimilarityThreshold()`

  Este componente requer um serviço que recupere o limiar de similaridade definido pelo usuário. Retorna um `Float` que indica o limiar de similaridade definido pelo usuário.

- **Eventos Anunciados**

  - `void knownUserFound(Long searchID, Integer userID)`

Este evento é anunciado sempre que é encontrado um dispositivo que pertence a um usuário que está na lista de contatos do usuário em questão. O parâmetro `searchID` indica o identificador da busca que encontrou o dispositivo e o parâmetro `userID` indica o identificador do usuário encontrado.

- `void similarUserFound(Long searchID, Integer userID, String userName, Set userInterests, Float similarity)`

  Este evento é anunciado sempre que é encontrado um dispositivo que pertence a um usuário que não está na lista de contatos do usuário em questão e cuja similaridade com o usuário em questão está acima do limiar de similaridade definido pela propriedade de inicialização `similarityThreshold`. O parâmetro `searchID` indica o identificador da busca que encontrou o dispositivo, o parâmetro `userID` indica o identificador do usuário encontrado, o parâmetro `userName` indica o nome do usuário, o parâmetro `userInterests` indica os interesses do usuário (um `Set` de palavras-chave, representadas por `String`) e o parâmetro `similarity` indica a similaridade existente entre os interesses do usuário em questão e do usuário encontrado.

- **Eventos de Interesse**

  - `void deviceFound(Long searchID, Integer ownerUserID, String userName, Set userInterests)`

    Este componente necessita receber este evento sempre que algum dispositivo se aproxima do dispositivo do usuário. O parâmetro `searchID` indica o identificador da busca que encontrou o dispositivo, o parâmetro `ownerUserID` indica o identificador do usuário que possui o dispositivo encontrado, o parâmetro `userName` indica o nome do usuário e o parâmetro `userInterests` indica os interesses do usuário (um `Set` de palavras-chave, representadas por `String`).

A.3 Componente `DeviceSearcher`

- **Descrição -** Busca dispositivos próximos ao dispositivo do usuário e anuncia a descoberta através do disparo de eventos.

- **Propriedades de Inicialização -** Não possui propriedades de inicialização.
- **Serviços Providos**
  - `Long searchNearbyDevices()`

    Busca dispositivos próximos ao dispositivo do usuário. Retorna um `Long` indicando o identificador da busca. Este identificador pode ser utilizado para interromper a busca, através do serviço `stopDeviceSearch`. Sempre que um dispositivo é encontrado através deste serviço, o evento `deviceFound` é anunciado.
  - `void stopDeviceSearch(Long searchID)`

    Interrompe a busca por dispositivos identificada por `searchID`.
- **Serviços Requeridos -** Não possui serviços requeridos.
- **Eventos Anunciados**
  - `void deviceFound(Long searchID, Integer ownerUserID, String userName, Set userInterests)`

    Este evento é anunciado sempre que o serviço `searchNearbyDevices` encontra um dispositivo. O parâmetro `searchID` indica o identificador da busca que encontrou o dispositivo, o parâmetro `ownerUserID` indica o identificador do usuário que possui o dispositivo encontrado, o parâmetro `userName` indica o nome do usuário e o parâmetro `userInterests` indica os interesses do usuário (um `Set` de palavras-chave, representadas por `String`).
- **Eventos de Interesse -** Não possui eventos de interesse.

A.4 Componente `ContentSharing`

- **Descrição -** Gerencia o acesso ao conteúdo compartilhado pelo usuário nas comunidades.
- **Propriedades de Inicialização**

  - `File contentFile`

    Esta propriedade de inicialização mantém referência para o arquivo-texto que armazena as informações do conteúdo disponibilizado pelo usuário nas comunidades.

- **Serviços Providos**

  - `Boolean addFile(File file, String interest, Integer role)`

    Adiciona o arquivo `file` à lista de arquivos disponibilizados na comunidade identificada por `interest`. O arquivo é disponibilizado apenas para os usuários que possuem o papel identificado pelo parâmetro `role`. Este serviço retorna `true` caso a lista de arquivos não possua o arquivo `file`; caso contrário, retorna `false`.

  - `File getFile(String fileName, Integer role)`

    Recupera o arquivo identificado pelo parâmetro `fileName`. O arquivo é retornado apenas se estiver disponibilizado para usuários que possuam o papel identificado pelo parâmetro `role`. Este serviço retorna o arquivo identificado pelo parâmetro `fileName`.

  - `Set getFilesByInterest(String interest, Integer role)`

    Recupera a lista de arquivos relacionados a um determinado interesse identificado pelo parâmetro `interest` e disponibilizados para os usuários que possuem o papel identificado pelo parâmetro `role`. Este serviço retorna um `Set` de `String`s com os arquivos recuperados.

- **Serviços Requeridos**

  - `Set getInterests()`

    Este componente requer um serviço que recupere a lista de interesses do usuário. Retorna um `Set` de `String`s com os interesses do usuário.

- **Eventos Anunciados -** Não possui eventos anunciados.

- **Eventos de Interesse -** Não possui eventos de interesse.

A.5 Componente `Similarity`

- **Descrição -** Calcula a similaridade entre usuários de acordo com os interesses de cada um deles.
- **Propriedades de Inicialização -** Não possui propriedades de inicialização.
- **Serviços Providos**
  - `Float evaluateSimilarity(Set userInterests, Set otherUserInterests)`

    Calcula a similaridade entre os interesses de dois usuários: `userInterests` e `otherUserInterests`. Ambos os interesses são definidos através de um `Set` de palavras-chave, representadas por `String`. Retorna um `Float` indicando a similaridade entre os interesses.
- **Serviços Requeridos -** Não possui serviços requeridos.
- **Eventos Anunciados -** Não possui eventos anunciados.
- **Eventos de Interesse -** Não possui eventos de interesse.

Apêndice B

Especificação dos Componentes do Estudo de Caso

Neste apêndice é apresentada a especificação do componente de interface gráfica desenvolvido especificamente para o estudo de caso apresentado neste trabalho. Os seguintes tópicos fazem parte da especificação do componente: descrição, propriedades de inicialização, serviços providos, serviços requeridos, eventos anunciados e eventos de interesse.

B.1 Componente `GUI`

- **Descrição -** Disponibiliza as informações e serviços da aplicação através de uma interface gráfica.
- **Propriedades de Inicialização -** Não possui propriedades de inicialização.
- **Serviços Providos -** Não possui serviços requeridos.
- **Serviços Requeridos**
  - `Long searchNearbyUsers()`

    Este componente requer um serviço que busque usuários próximos a um determinado usuário. Retorna um `Long` indicando o identificador da busca. Este identificador poder ser utilizado para interromper a busca, através do serviço requerido `stopUserSearch`.

- `void stopUserSearch(Long searchID)`

  Este componente requer um serviço que interrompa a busca por usuários identificada por `searchID`.

- `Set getInterests()`

  Este componente requer um serviço que recupere a lista de interesses do usuário. Retorna um `Set` de `String`s com os interesses do usuário.

- `Boolean addInterest(String interest)`

  Este componente requer um serviço que adicione o interesse `interest` à lista de interesses do usuário. Este serviço retorna `true` caso a lista de interesses do usuário não possua o interesse `interest`; caso contrário, retorna `false`.

- `Boolean removeInterest(String interest)`

  Este componente requer um serviço que remova o interesse `interest` da lista de interesses do usuário. Este serviço retorna `true` caso a lista de interesses do usuário já possua o interesse `interest`; caso contrário, retorna `false`.

- `Set getContacts()`

  Este componente requer um serviço que recupere a lista de contatos do usuário. Retorna um `Set` de `Contact`s com os contatos do usuário.

- `Boolean addContact(Contact contact)`

  Este componente requer um serviço que adicione o contato `contact` à lista de contatos do usuário. Este serviço retorna `true` caso a lista de contatos do usuário não possua o contato `contact`; caso contrário, retorna `false`.

- `Float getSimilarityThreshold()`

  Este componente requer um serviço que recupere o limiar de similaridade definido pelo usuário. Retorna um `Float` que indica o limiar de similaridade definido pelo usuário.

- `void setSimilarityThreshold(Float similarityThreshold)`

  Este componente requer um serviço que defina o limiar de similaridade do usuário de acordo com o valor do parâmetro `similarityThreshold`.

- `Map getCommunities()`

  Este componente requer um serviço que recupere a lista de comunidades do usuário. Retorna um `Map` cuja chave é uma `String` que representa um interesse e cujo valor é um `Set` de `Contacts` com os contatos associados ao interesse em questão.

- `Boolean addFile(File file, String interest, Integer role)`

  Este componente requer um serviço que adicione o arquivo `file` à lista de arquivos disponibilizados na comunidade identificada por `interest`. O arquivo deve ser disponibilizado apenas para os usuários que possuem o papel identificado pelo parâmetro `role`. Retorna `true` caso a lista de arquivos não possua o arquivo `file`; caso contrário, retorna `false`.

- `File getFile(String fileName, Integer role)`

  Este componente requer um serviço que recupere o arquivo identificado pelo parâmetro `fileName`. O arquivo é retornado apenas se estiver disponibilizado para usuários que possuam o papel identificado pelo parâmetro `role`. Retorna o arquivo identificado pelo parâmetro `fileName`.

- `Set getFilesByInterest(String interest, Integer role)`

  Este componente requer um serviço que recupere a lista de arquivos relacionados a um determinado interesse identificado pelo parâmetro `interest` e disponibilizados para os usuários que possuem o papel identificado pelo parâmetro `role`. Retorna um `Set` de `Strings` com os arquivos recuperados.

- **Eventos Anunciados -** Não possui eventos anunciados.

- **Eventos de Interesse**

  - `void similarUserFound(Long searchID, Integer userID, String userName, Set userInterests, Float similarity)`

    Este componente requer um evento que seja anunciado sempre que é encontrado um dispositivo que pertence a um usuário que não está na lista de contatos do usuário em questão e cuja similaridade com o usuário em questão

está acima do limiar de similaridade definido pela propriedade de inicialização `similarityThreshold`. O parâmetro `searchID` indica o identificador da busca que encontrou o dispositivo, o parâmetro `userID` indica o identificador do usuário encontrado, o parâmetro `userName` indica o nome do usuário, o parâmetro `userInterests` indica os interesses do usuário (um `Set` de palavras-chave, representadas por `String`) e o parâmetro `similarity` indica a similaridade existente entre os interesses do usuário em questão e do usuário encontrado.